A MARSHALL CAVENDISH 19 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95

INPUT

Vol. 2 **No 19**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS

Front cover, NASA/Bernard Fallon. Page 573, Dave King/Studio 10. Page 574, Dave King/Steve Cross. Pages 578, 580, 581, 582, 583, Kevin O'Keefe. Page 584, 586, 589, 590, Paul Chave/Bernard Fallon. Pages 587, 592, Bernard Fallon. Pages 593, 594, Graeme Harris. Page 596 Mickey Finn. Pages 598, 600, 602, 604, Gary Wing.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland: Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd. Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

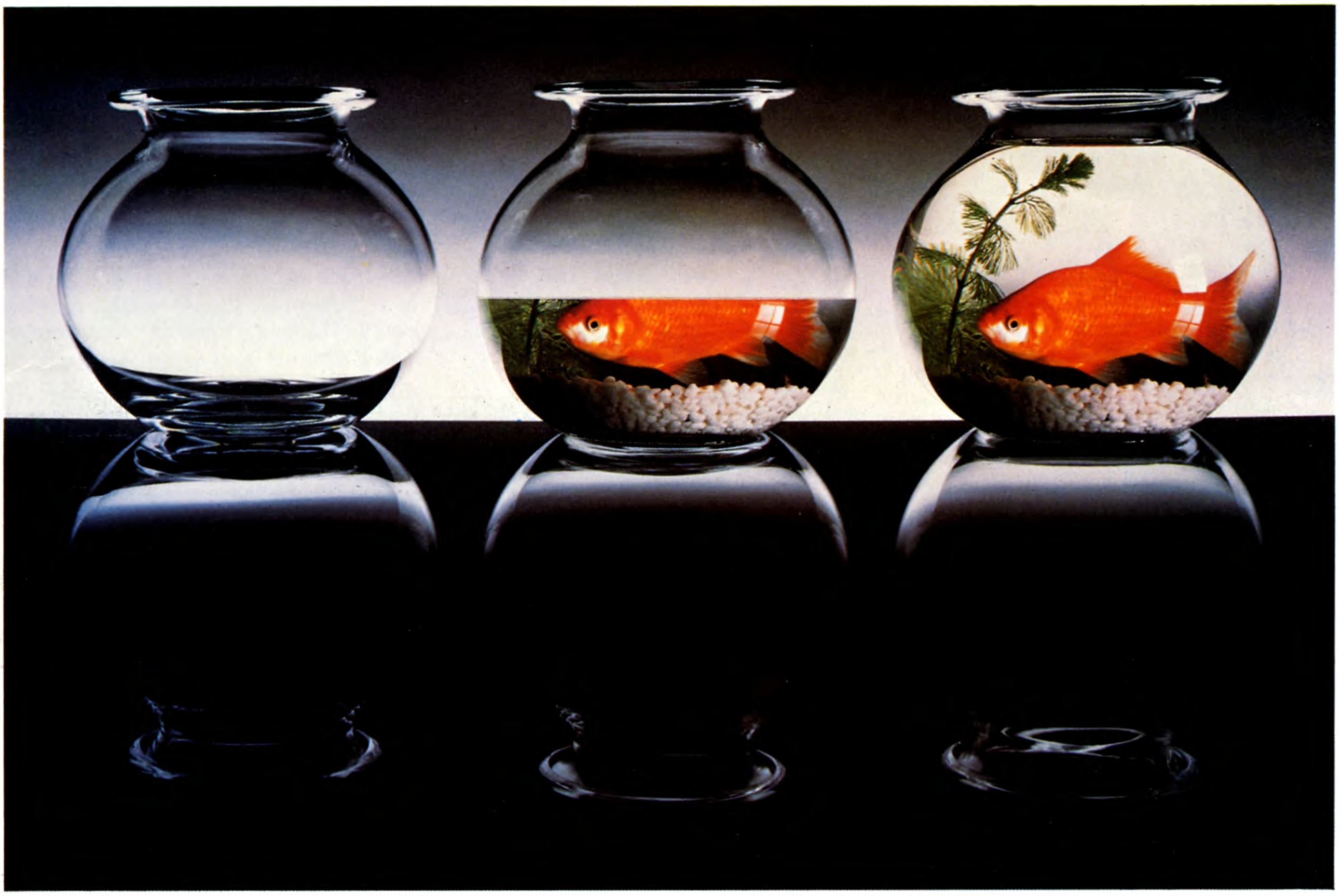
The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

COMPUTER AIDED DESIGN-2

Add these routines to your CAD (Computer Aided Design) listing to improve its sophistication and give an enormous boost to your drawing and design skills



The Computer Aided Design (CAD) program on page 566 showed how you can put your computer's high-resolution graphics under keyboard control and allowed you to construct some sophisticated outline drawings. But there are several functions still required to give the program its full potential—notably the ability to fill in with colour. You'll have seen these additional options on offer on the menu, but as yet, you won't have been able to access them.

LOAD the program from the first article, then key in the additional lines given here. There is of course no section for the Vic 20, because the entire program was given in the first article. Each section of program is followed by notes on using the new facilities and there is a further guide for Vic users.

```
4000 REM rectangle and box
4010 LET box=Ø: GOTO 4030
4020 LET box=1
4030 FOR n=1 TO 5Ø: NEXT n
4040 GOSUB 8000
4050 IF INKEY$=CHR$ 13 THEN
  RANDOMIZE USR 6538Ø: GOTO 1ØØØ
4060 IF INKEY$<>CHR$ 32 THEN GOTO
  4Ø4Ø
4070 FOR n=1 TO 5Ø: NEXT n
4080 LET xx=Ø: LET yy=Ø: LET hx=x: LET
  hy=y
4090 GOSUB 8ØØØ: FOR n=1 TO 2: PLOT
  hx,hy
4100 DRAW OVER 1;Ø,yy: DRAW OVER
  1;xx,Ø: DRAW OVER 1;Ø,-yy: DRAW
  OVER 1;-xx,Ø: NEXT n
4110 IF INKEY$<>CHR$ 32 THEN GOTO
  4Ø9Ø
4120 PLOT hx,hy: DRAW Ø,yy: DRAW xx,Ø:
  DRAW Ø,-yy: DRAW -xx,Ø
4130 IF box=Ø THEN GOTO 4Ø3Ø
4135 IF xx=Ø THEN GOTO 4Ø4Ø
4140 FOR n=hx TO hx+xx STEP SGN xx
4150 PLOT n,hy: DRAW Ø,yy: NEXT n
4160 GOTO 4Ø4Ø
5000 REM circle
5010 FOR n=1 TO 5Ø: NEXT n
5020 GOSUB 8ØØØ
5030 IF INKEY$=CHR$ 13 THEN
  RANDOMIZE USR 6538Ø: GOTO 1ØØØ
5040 IF INKEY$<>CHR$ 32 THEN GOTO
  5Ø2Ø
5050 FOR n=1 TO 5Ø: NEXT n
```

```
5060 LET xx=0: LET yy=0: LET hx=x: LET
   hy=y
5070 GOSUB 8000: CIRCLE OVER
   1;hx,hy,ABS xx: CIRCLE OVER 1;hx,hy,ABS
   xx
5080 IF INKEY$<>CHR$ 32 THEN GOTO
   5070
5090 CIRCLE hx,hy,ABS xx:
   GOTO 5000
5500 REM erase
5510 GOSUB 8000
5520 IF POINT (x,y)=1 THEN PLOT OVER
   1;x,y
5530 IF INKEY$=CHR$ 13 THEN
   RANDOMIZE USR 65380:
   GOTO 1000
5540 GOTO 5510
6000 REM copy
6010 COPY : GOTO 1000
7000 INPUT "ENTER NAME□"; LINE n$: IF
   LEN n$>10 THEN GOTO 7000
7010 LOAD n$CODE 50000: RANDOMIZE
   USR 65368: GOTO 1000
7500 INPUT "ENTER NAME□"; LINE n$: IF
   n$="" OR LEN n$>10 THEN GOTO
   7500
7510 SAVE n$SCREEN$ : GOTO 1000
```

Select the RECTANGLE option, and move the cursor to one corner of the rectangle you wish to draw, then press [SPACE]. Move to the diagonally opposite corner. As you move, the rectangle will flash on the screen, so you can adjust its size and shape merely by moving the cursor. When you are satisfied, press [SPACE] again to fix its position.

The BOX option is as RECTANGLE, except that the area inside is filled.

CIRCLE is the other shape you can draw. Move the cursor to where you wish the centre of the circle to be, then press [SPACE]. Next move to any point on the circumference of the circle and press [SPACE] again.

The COPY option lets you send a screen image to a ZX printer. Select the option and respond to the prompts on the screen. After copying, the display returns to the menu.

To correct errors or change areas of the image, use either ERASE or OOPS. You can erase small detail by moving the cursor over it to remove a pixel at a time. After erasing, press [ENTER] to return to the menu.

For more extensive changes, select OOPS. This will automatically clear everything you have done since you last visited the menu.

The two last items on the menu are SAVE and LOAD (to or from tape only). When you select either option, you are prompted to supply a file name. You can LOAD without specifying a file name, by pressing [ENTER], but you must supply a name to SAVE.

```
170 IF A$=CHR$(135) THEN 540
180 IF A$=CHR$(136) THEN 590
190 IF CO<4 AND A$="P" THEN PAINT
   X,Y+1,CO
260 IF A$="R"THEN F=6:XX=X:
   YY=Y
280 IF CO<4 AND F=6THENLINE
   XX,YY,X,Y,CO:CO=4
300 IF A$="B" THEN XX=X:
   YY=Y:F=1:CO=4
320 IF (F=1 AND A$="□") AND CO<5
   THEN GOSUB 490
330 IF A$="E" THEN XX=X:
   YY=Y:F=2:CO=4
340 IF (F=2 AND A$="□") AND CO<4
   THEN CIRCLEXX,YY,ABS
   (X-XX),ABS(Y-YY),CO:F=0
370 IF A$="C" THEN XX=X:
   YY=Y:F=4:CO=4
380 IF (F=4 AND A$="□") AND CO<4
   THEN CIRCLEXX,YY,ABS
   (X-XX),ABS(X-XX),CO:F=0
410 IF A$<>"T" THEN 440
420 GET A$:IFA$="" THEN 420
430 TEXT X,Y,A$,4,1,1
490 IF YY>Y THEN YT=YY:
   YY=YY-ABS(YY-Y):Y=YT
500 IF XX>X THEN XT=XX:
   XX=XX-ABS(XX-X):X=XT
510 BLOCK XX,YY,X,Y,CO:F=0:
   CO=4:RETURN
540 NRM:NM$="":INPUT "♥ENTER NAME
   TO SAVE";NM$:IF NM$="" THEN 130
550 GOSUB 610:IF IN$="D" THEN
   NM$="@:"+NM$
560 POKE 24432,LEN(NM$):POKE 24388,DV
570 FOR Z=1 TO LEN(NM$):POKE
   24432+Z,ASC(MID$(NM$,Z,1)):NEXT
580 SYS 24379:GOTO 130
590 NRM:NM$="":INPUT "♥ENTER NAME
   TO LOAD";NM$:IF NM$="" THEN 130
600 GOSUB 610:LOAD NM$,DV,1:END
610 IN$="":INPUT "♥(T)APE OR
   (D)ISC";IN$
620 IF IN$="D" THEN DV=8:
   RETURN
630 IF IN$="T" THEN DV=1:
   RETURN
640 GOTO 610
```

The earlier part of the program includes the Draw facility, with the step options, to draw polygons, such as triangles, rectangles and pentagons. But now you can obtain these straight line shapes better if you use Line, instead. To draw a triangle, for example, move the cursor to where you wish one of the corners to be and press [L]. Next, move the cursor to the second corner, select a colour in which to draw and press the space bar. Without moving the cursor, press [L] again to fix the start of the next line, then move to the third corner. Select a colour and fix the line as before, then draw the third.

The line flashes as you move (until you select a colour), so it is simple to plan the size and shape of the triangle, or any polygon.

The Circle option is given by [C]. Move to where you wish the centre of the circle to be and press [C]. Now move to a point on the circumference of the circle (you can move only horizontally or vertically). When you are satisfied with the size of the circle, select a colour for the circle and press the space bar.

If you are used to the CIRCLE command on the Commodores, you will not be surprised that the shape is not circular, but elliptical. In fact, to draw true circles, as well as ellipses, you need to use Ellipse. Move to the centre of the ellipse—where the two axes meet—then press [E]. Move away from this point so that its vertical distance from the cursor is half one axis, and its horizontal distance is half the other. Now select a colour and press the space bar to draw the ellipse. After a few attempts, you will be able to judge the length each axis should be so that you get any ellipse or circle.

Specifying the size of the ellipse may take a little getting used to at first. Suppose you want an ellipse that is 150 mm long and 100 mm high. Move the cursor away from the centre half the length of each of these axes—75 mm to left or right and 50 mm up or down.

The Radial lines facility goes well with Circle or Ellipse. Draw either of these shapes, then move to the centre and press [R]. Now you can move to any point on the circumference, select a colour and press the space bar to fix a radius—a line between the centre point and the new cursor position. Without moving back to the centre, you can fix any number of radii, each to the first point.

The last of the shapes in this program is Block, which draws a filled-in rectangle. Move to any point on the screen and press [B] to fix one corner of the block. Move to another point to fix the diagonally opposite corner, then select a colour and press the space bar.

If you wish to colour any enclosed shape on the screen, then Paint is useful. Move to any point within the shape, then select a colour. Now press [P] to colour the shape.

Adding text to your graphics is a simple matter. Move to where you wish to place each character and press [T], then key the character. It will be printed in the inverse of the colour on the screen. Repeat for each character.

You have a choice of methods for correcting errors. One of these is to press [SHIFT] and [CLR/HOME] together, when the screen clears



completely and you can start afresh. A less drastic method is to delete an area of screen, using Block in the background colour. To delete even smaller areas—a line or point, say— use Draw in the background colour.

A more powerful method of correcting errors is to press key [f1] while drawing. This copies the image at that moment to a section of memory. If you make an error, or are dissatisfied with what you have drawn since pressing [f1], press [f3] to recall the previous screen you stored. When using the program for the first time, press key [f1] to prepare the area of memory used to store the image.

Press [f1] then [f5] to SAVE the image you have drawn on to tape or disk. To use this facility, press the key and respond to the prompts on the screen. When you are asked for a file name, you can abort the saving routine by pressing [RETURN].

To LOAD an image from disk or tape, press [f7], then respond to the prompts. After loading, press [f3] to call up the image.

The program appeared with the first part of this article, but some of its facilities have not been examined in detail. You may already have worked out one of the two methods of correcting errors. To correct small areas, select the background colour (1) and move the cursor to draw over the points to be deleted. Only if the image you have drawn is unsatisfactory, or no longer needed, will you wish to use the second method—to delete the entire screen. You can do so by pressing [SHIFT] and [CLR/HOME] together.

To draw outline shapes, such as rectangles and triangles, use the variable step facility. If you select the normal step (key 5), you can move with the control keys to draw a rectangle, for example, in which each side is made up of a large number of lines. As you select larger steps, you will be able to draw a rectangle in which each side is made up of a single line. Select the step, move (holding [SHIFT]) to the first corner of the rectangle, then draw the first side, by pressing the control key for the direction in which you wish to draw. Release the control key when the side is the length you require. Press another control key to draw an adjacent side, and so on, until the fourth side is drawn.

Alternatively, use Line to draw straight lines in any direction. Move the cursor to where you wish the line to start, then press the arrow key at the top, left-hand corner of the keyboard to fix the cursor position. Move to where the line should end, then press [RETURN] to draw the line. If you move to another position and press [RETURN], a line will be drawn from the original starting position to this point.

Using this method, you can draw any number of radial lines—to form the spokes of a wheel, for example. If you wish to draw a line from a different starting position, move to the new point, then press the arrow key.

To draw circles and ellipses, move to a point and press the arrow key to fix the centre. Move the cursor so that its distance above or below the centre is half one axis of the ellipse, and its distance to the left or right of the centre is half the other axis. (If you find this confusing, there is a fuller explanation in the Commodore 64 section). Now press [C] to draw the ellipse. When both axes are equal, the shape should be a circle, but it does not appear so—due to the asymmetrical arrangement of the pixels. But with practice, you will be able to judge the length of each axis so that a circle is drawn.

The last facility of this program lets you fill enclosed areas of screen. Simply move to a point within the area, select a colour (1 to 4), then press [F] to fill it in.

```
660 IF NOT INKEY-99 THEN ENDPROC
670 MOVEX2,Y2:MOVEX3,Y3
680 PLOT85,X,Y
690 X3=X2:X2=X:Y3=Y2:Y2=Y
700 IF INKEY-99 THEN 700
730 VDU5
740 MOVE X,Y+32
750 I=INKEY(1)
760 IF I=13 THEN VDU4:A2$=
    A3$:ENDPROC
770 IF I<32 OR I>127 THEN 750
780 VDU I
790 IF I=127 THEN X=X-32: GOTO 750
800 X=X+32:GOTO 750
840 IF NOT INKEY -99 THEN ENDPROC
850 XR=X2-X:YR=Y2-Y
860 MOVE X2+XR,Y2
870 FOR T=0 TO 2*PI+.02 STEP .05
880 DRAW X2+XR*COS(T),Y2+YR*SIN(T)
890 NEXT
900 IF INKEY -99 THEN 900
980 TC=C:C=GET-48
990 IF C>9 THEN C=C-7
1000 IF C<0 OR C>MC THEN 1040
1010 GCOL0,C
1020 PRINTTAB(20,0)"COLOUR□";
     C;"□"
1030 GOTO 1060
1040 C=C+55:IF C>129 AND C<140
     THEN A3$=CHR$(C)
1050 C=TC
1060 A2$=A3$
1090 IF NOT INKEY -99 THEN ENDPROC
1100 MOVE X,Y2:PLOT 85,X2,Y:
     PLOT 85,X2,Y2
1110 IF INKEY-99 THEN 1110
1120 X3=X2:X2=X:Y3=Y2:Y2=Y
1150 IF NOT INKEY -99 THEN ENDPROC
1160 DRAWX,Y2:DRAWX2,Y2:
     DRAWX2,Y:DRAWX,Y
1170 IF INKEY-99 THEN 1170
1180 X3=X2:X2=X:Y3=Y2:Y2=Y
1205 SOUND1,-15,0,10
1210 *LOAD"PIC"
1215 SOUND1,-15,150,20
1255 SOUND1,-15,0,10
1260 ON M+1 GOTO 1270,1270,
     1270,0,1300,1300
1270 *SAVE"PIC" 3280 8000
1280 GOTO 1305
1300 *SAVE"PIC" 5940 8000
1305 SOUND1,-15,150,20
```

SAVE the new program in case of mishaps, then RUN it to test your new options.

To select a colour in which to draw, press [f6], when 'Change colour' will appear on the screen. Enter a value from 0 up to the maximum permitted for the mode selected. If you selected mode 2, where you have a choice of 16 colour effects, 0 to 9 give the first ten colour effects and A to F give the next six. The number of the colour you select appears after 'COLOUR' at the top of the screen. Then the display will show the drawing option you were using before you pressed [f6] (in this case Point).

The movement of the cursor, either with or without the Draw option, can be speeded up by a factor of two, four or eight. Hold down [CTRL] while pressing any of the arrow keys to give twice normal speed, [SHIFT] for four times normal speed or [CTRL] and [SHIFT] together for eight times normal speed. If you use Draw at more than normal speed, then a dotted line will be drawn—the faster the speed, the farther apart the dots appear.

The Triangle option is selected by [f2]. First select Point, [f0], and establish two of the corners of the triangle. If you don't set new points, then the last two points visited by the cursor will be selected. Then change the drawing colour, if you wish, then select Triangle. Move to the third corner of the triangle, and press the space bar to fill a triangular area of the screen.

To use the Text option, move the cursor to where you wish the bottom left of the first character to appear and press [f3]. Key your text, then press [RETURN], when the display will return to the previous drawing option.

To use the Ellipse option, [f4], select [f0] and establish a centre point about which to draw the ellipse. If you do not wish a dot to appear on the screen, select colour 0 before

pressing the space bar to establish the centre. Now move the cursor away from this point to fix the size and shape of the ellipse. The distance of the cursor above or below the starting point is half the height of the vertical axis, and the distance to the left or right of it is half the width of the horizontal axis. If the distances in both directions are equal, then the shape will be a circle. Change the colour to the one you want then press the space bar to draw the ellipse.

The Box option, f7, lets you fill a rectangular area of screen. Move to one corner and press the space bar while in point mode f0, then move to the diagonally opposite corner and press again in box mode f7.

The Frame option, f8, works as Box, except that you only get the outline.

There are no delete options, but you can use the Change colour option, f6, together with the drawing options, such as Point, Line, Draw and Triangle, to delete either small detail (using Point) or larger areas of the screen. To delete a rectangular area, say, set colour 0, then fill it, using Point and Frame.

The last two facilities offered by this program are SAVE and LOAD, using either a tape or a disk system. To save a screen, do not End by pressing f9. Instead, prepare the disk, or set the tape recorder to record. Now press CTRL and S, when a low pitched beep will sound to signify saving has begun. When saving is complete, a high pitched beep will sound. Apart from this, you cannot check that saving is satisfactory, so ensure your volume and tone controls are adjusted properly.

To Load the screen back into memory, press CTRL and L (a low toned beep will sound), then play the tape.

```
4000 SCREEN1,ST:GOSUB1500
4010 IFEF=1 GOSUB500:RETURN
4020 IFPEEK(345)<>PC THEN4000
4030 XS=X:YS=Y
4040 GOSUB1500:GOSUB500
4060 IFEF=1 THENRETURN
4070 IF ABS((XS-X)*(YS-Y))>
   23000 THEN 4040
4080 LINE(X,Y)-(XS,YS),PSET,B
4090 IFPEEK(345)<>PC THEN 4040
4100 PMODEMD,5:GET(X,Y)-(XS,YS),
   CP,G:PMODEMD,1
4110 XS=XS-X:YS=YS-Y
4120 GOSUB1500:GOSUB500
4130 IFEF=1 THENRETURN
4140 IF(X+XS)<0OR(X+XS)>255
   OR(Y+YS)<0OR(Y+YS)>191
   THEN4120
4150 LINE(X,Y)-(X+XS,Y+YS),
   PSET,B
4160 IFPEEK(345)<>PC THEN4120
4170 GOSUB500:PUT(X,Y)-(X+XS,
   Y+YS),CP,PSET:GOSUB510
4180 GOTO4120
5000 CLS:PRINT"□SELECT THE BORDER
   COLOUR (0-8) ?"
5010 A$=INKEY$:IF A$<"0"OR
   A$>"8"THEN5010
5020 BC=VAL(A$):SCREEN1,ST
5030 GOSUB1500:GOSUB500
5040 IF EF=1 THENRETURN
5050 IFPEEK(345)<>PC THEN5030
5060 PAINT(X,Y),CL,BC:GOSUB510:
   GOTO5030
6000 CLS:PRINT"□SAVE OR LOAD FROM
   TAPE (S/L) ?"
6010 A$=INKEY$:IFA$<>"S"AND
   A$<>"L"THEN6010
6020 IF A$="S" THEN 6100
6030 PRINT"□ARE YOU SURE YOU WISH
   TO LOAD□□□ANOTHER PICTURE
   (Y/N) ?"
6040 A$=INKEY$:IF A$<>"N"AND
   A$<>"Y" THEN 6040
6050 IF A$="N" THEN RETURN
6060 MOTORON:PRINT:PRINT
   "□POSITION TAPE, PRESS PLAY AND
   □□□THEN PRESS ENTER."
6070 IFINKEY$<>CHR$(13)THEN6070
6080 MOTOROFF:PRINT:INPUT
   "INPUT THE FILE NAME□";A$
6090 SCREEN1,ST:CLOADM A$:
   GOSUB510:RETURN
6100 PRINT:INPUT"KEY FILE NAME□";A$
6110 MOTORON:PRINT:PRINT
   "□POSITION TAPE, PRESS RECORD AND
   THEN PRESS ENTER."
6120 IFINKEY$<>CHR$(13)THEN6120
6130 GOSUB500:CSAVEM A$,1536,
   7679,1536:RETURN
```

Microtip

Spectrum Colours

You can use the complete listing to do colour drawings on the Spectrum micro, but take care to avoid problems of overlapping colours. If you place more than two colours on each character square, the last colour will overwrite the previous one. Plan your image so that adjoining colours are aligned with the character squares. As a guide, these are in 32 blocks of eight squares across and 22 blocks of eight squares from top to bottom of the screen.

The Rectangle option lets you draw rectangles in various colours at any place on the screen. Select the option, then move to a point on the screen where you want one corner of the rectangle to be. Press the space bar to select this point. Now move the cursor to the diagonally opposite corner of the rectangle, and watch the shape change as you move. As with Draw and Line, you can select colours, or abort (by pressing ENTER). When you are satisfied with the shape and size of the rectangle, press the space bar to draw it.

Box works just as Rectangle, except that the interior is filled with the colour in use.

Circle and Disc are like Rectangle and Box: the first point you select lies on the circumference. Then you move the cursor, noting the size of the shape, to the second point, which is at the centre. Disc is a filled in circle.

In use, Ellipse is slightly different from Circle and Disc. The first point you select is the centre. As you move the cursor either up and down or left and right away from the centre, you will see only a straight line, but if you move it right, then up or down, or left then up or down, it starts to grow an ellipse. Once you have selected the centre, move in various directions away from the centre and you will see an ellipse flashing on the screen, then press the space bar to fix it.

The Copy option lets you duplicate the image that you have already drawn on one part of the screen, on to another part of the screen. To use this option, move the cursor to one corner of the area to be copied, then press the space bar. Now as you move the cursor to the opposite corner, you will see a rectangle or square flash on the screen. Select the area (by pressing the space bar) when you are satisfied with its size and shape. Move the area to the new site (using the arrow keys) and press the space bar to copy it, blotting out what was already on the screen. You can copy as much as a half of the screen, but the area to be copied must be totally within the screen.

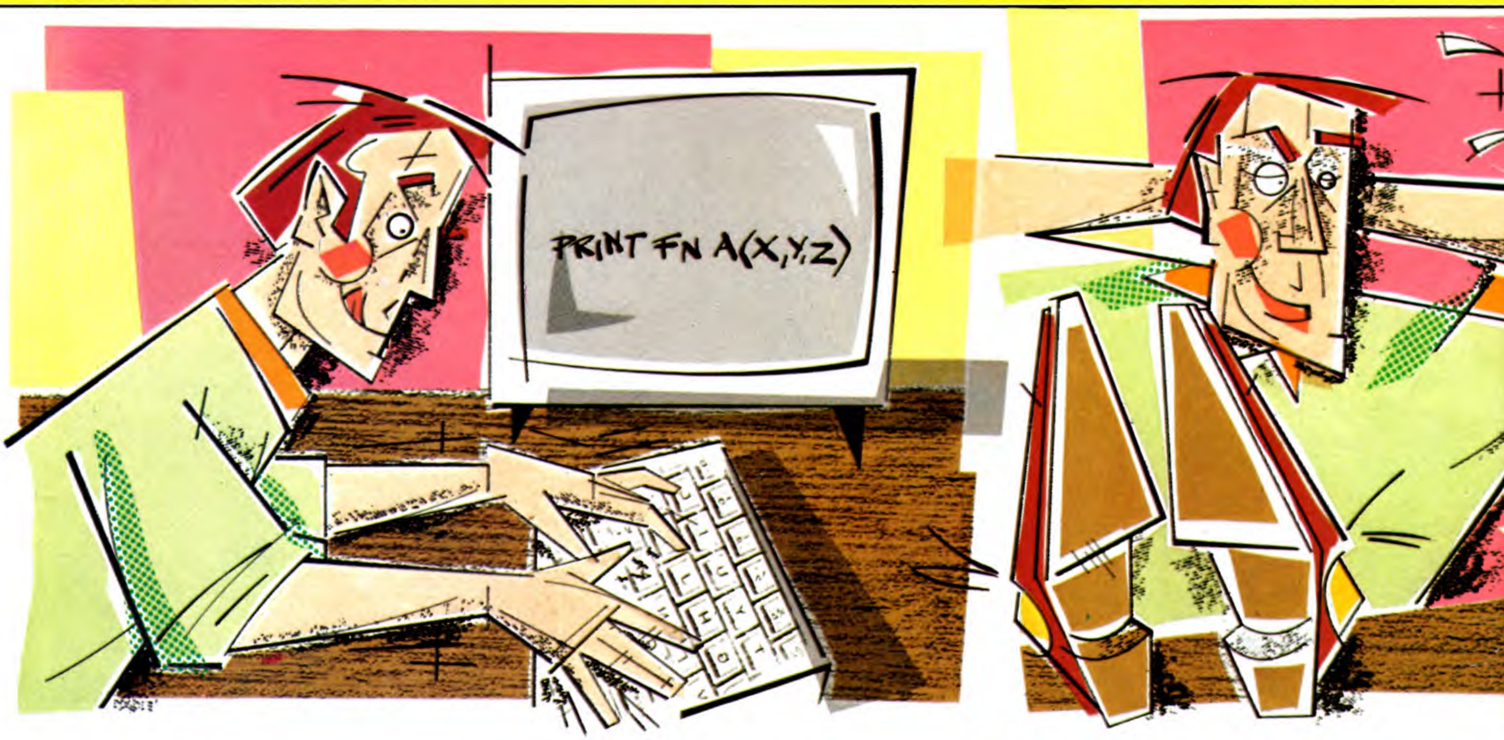
The Fill option works just as the BASIC PAINT command. When you select it, you are asked to supply the colour at which the PAINT will stop. You should then move to the area to be filled and press the space bar. The PAINT colour can be changed, using keys 0 to 8.

To correct your mistakes, there is an Error option, which erases everything you have drawn since your last visit to the main menu. You can, however, erase fine detail by first selecting the background colour. In fact, you can use this method with Box or Disc to erase larger areas of the screen image.

The last option lets you Load or Save an image with a tape system: connect the recorder and respond to the prompts on the screen.

CREATING CUSTOM FUNCTIONS

There are no BASIC functions for things like cubes, compound interest, or capital letters. But many computers enable you to create special functions when needed



Your computer can do a great number of things—especially those involving mathematical calculations—much faster than you can yourself. But it can only do what it is told: either by you, in a program that you type in yourself, or by someone else, whose program you LOAD into your computer.

The most common language that people use to program their micro is BASIC, the language into which all the computers covered here automatically go at switch-on. And your computer has various functions built into its standard BASIC: functions like SIN, COS, SQR and many others. Each of these is defined in such a way that your computer knows how to recognize the command and perform the appropriate operation on a given value—calculating its sine, cosine, square root, or whatever.

BASIC FUNCTIONS

This is all very well—so long as the functions you want to use in your program are also BASIC keywords for your computer. But many common functions are not part of the standard BASIC—there is no function called CUBE for example, which will automatically tell you the value of x*x*x. If a function you want to use frequently is not available, you have a number of options. You can devise a way of avoiding the use of the function in your program; you can add a subroutine which works out the function that you want; or, most elegantly of all, you can employ the user-definable function facility of your computer. This option is not open to users of the ZX81. But on the Spectrum, BBC B and Electron it is often a very neat way around a problem. The Commodore 64, Vic 20, Dragon and Tandy also have the facility, although it is rather more limited.

User-definable functions literally give you the capacity to tailor your computer's BASIC to suit the needs of a particular program. The command which actually defines the function is slightly different for each computer, but takes this basic form:

DEF FN a (x,y) = . . . (whatever you want the function to do)

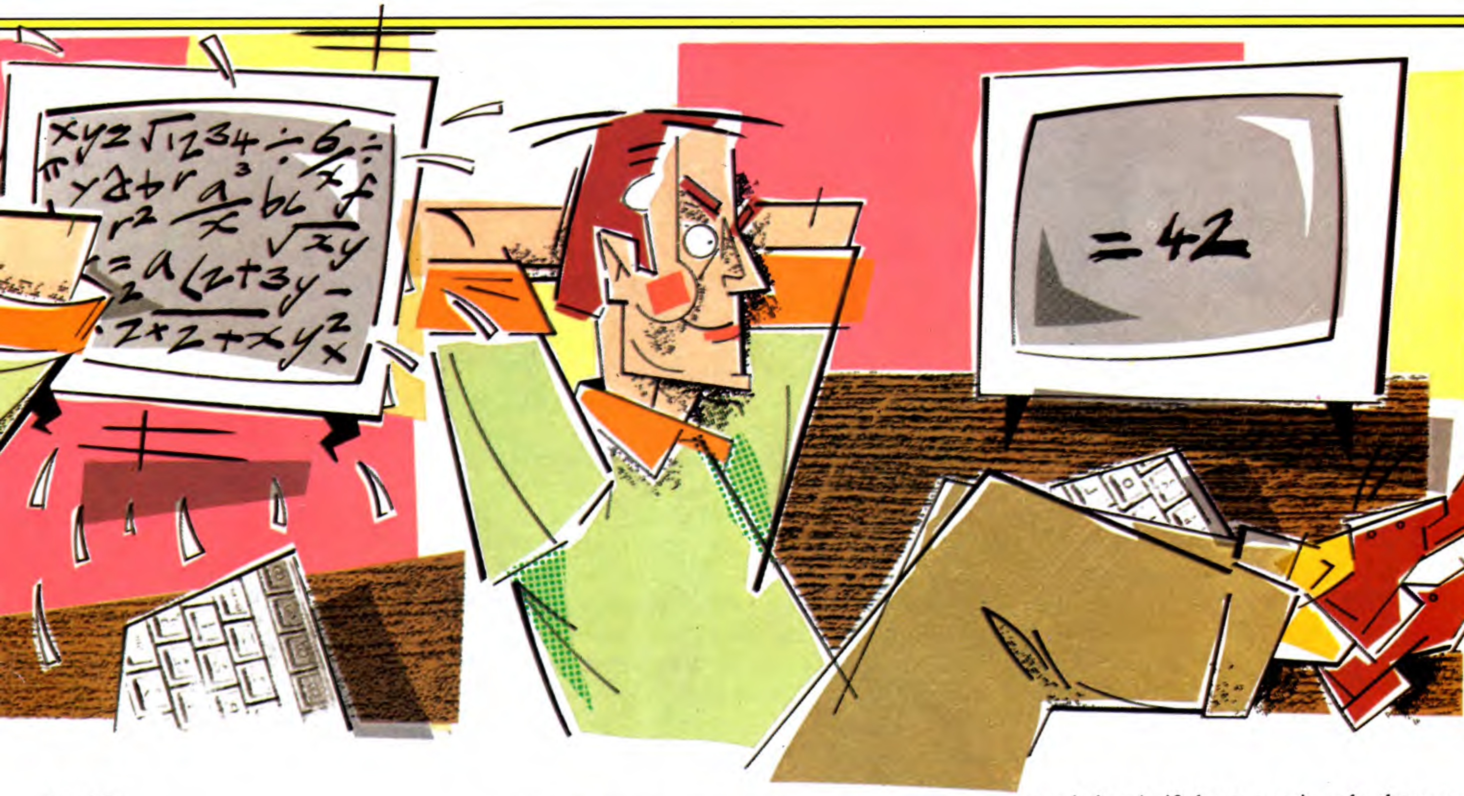
Here, the letter a is the name of the function (which you will need in order to call it up) and the letters in brackets are the parameters which the function uses but there are not necessarily only two of them. The Spectrum and Acorn can have several, but the Commodore 64, Vic 20, Dragon and Tandy can only use one parameter in each function.

To see how the general principle works in an actual program, type in and RUN this example. This defines the very simple function mentioned above—it gives you the cube of a number.

```
10 CLS
20 PRINT "NUMBER","CUBE"
30 FOR a=1 TO 20
40 PRINT a,FN c(a)
50 NEXT a
60 DEF FN c(x)=x*x*x
```

```
20 CLS
30 PRINT"NUMBER","CUBE"
40 FOR A=1 TO 13
50 PRINTA,FNC(A)
60 NEXT
```

- BASIC FUNCTIONS
- HOW TO DEFINE YOUR OWN FUNCTIONS
- WHAT TO CALL A FUNCTION
- HOW MANY PARAMETERS?
- USING A FUNCTION
- USING FUNCTIONS TO LIVEN UP INPUT STRINGS
- DRAWING ELLIPSES
- INTERESTING FUNCTIONS




```
70 END
80 DEF FNC(X)
90 =X*X*X
```

```
10 DEF FNC(X)=X*X*X
30 PRINT"NUMBER","CUBE"
40 FOR A=1 TO 13
50 PRINTA,FNC(A)
60 NEXT
```

An interesting point to note is that in the Acorn and Spectrum programs, the computer never actually reaches the program lines which DEFine the FunctioN. In this way, they are like DATA statements: the computer uses them without actually having to RUN through them. This is not true for the Commodore 64, Vic 20, Dragon and Tandy, which must read the DEF lines. The Spectrum searches for the DEF statements starting at the beginning of the program, each time the function is called. So in a long program it is best to group all the DEF FN lines at the start. The Acorns' DEF FN lines must not be RUN and should be grouped after END.

CALLING A FUNCTION

You call a function quite simply: by placing the keyword FN and the name of the function wherever you want the number which results from the function. You can see this from Line 40 in the Spectrum program, Line 50 in the Acorn program, and Line 50 in the Commodores, Dragon and Tandy version. In the given example, the program uses FNC(A)—notice that Spectrum syntax requires the form FN c(a)—where A is the number you want cubed. If you wanted to design your own program to use this function, you would just use FNC(A), or FN c(a), anywhere you would otherwise have put a*a*a.

The same principle applies to any other use of a defined function—the call can effectively be treated as a form of shorthand for the calculation itself, whatever it may be.

For such a simple function as the example above, you might think it is quicker to use the power function already built into your computer's BASIC. In fact, even in this example it is actually quicker to call the function (or to multiply the number by itself several times), if the power is only three or less. And of course, DEF FN is capable of handling far more complex calculations than this example.

Although you would not normally want to use a function for this simple type of calculation, you can learn how DEF FN works on your computer by experimenting with your own changed versions of the last program. Try modifying it to define new functions to divide numbers by 10, multiply by two thousand or work out the SINe. Then use the maths functions which your computer already has in its BASIC. If you want to check that you have defined the function correctly you can compare it with what you get when the computer works out the answer as a direct command.

While experimenting, it is useful to understand how DEF FN actually works on your computer.

FUNCTION NAMES

Each function you define has to have a name that you use to call the function for use in your programs.

The name is a letter, or collection of letters, which is placed immediately after the DEF FN like this: DEF FNacorn (which starts to define a function called acorn). The Spectrum name must be just one letter. The Commodores can have one or two.

The Acorn has no real limit on how many letters this name can contain: if you want, you can DEFine a FunctioN with a name hundreds of letters (or numbers) long.

The same is also true of the Commodores, Dragon and Tandy, but they only count the first two characters. The first character of each name, though, must be a letter.

So although names such as ACORN would be allowed, the Commodores, Dragon and Tandy would not be able to tell the difference between it and 'ACTOR' or any other name beginning AC. The Acorn, on the other hand, would differentiate between the two.

DEFINING PARAMETERS

The next step in defining your function is adding the 'parameters'. These are the numbers or letters in brackets after the name of the function. While they may look like variables, they are not (except on the Commodores) but they are so similar that they are sometimes called 'dummy variables'.

All they do is tell the computer that the function will use that many numbers from the program in its calculations. In other words, if you start off your DEF FN like this:

```
DEF FNexample (a,b,c)
```

then the computer expects three numbers from the program itself. So, when calling or using the function in your program you would include three numbers, or variables, in brackets (how you actually use your defined functions is explained below).

While the parameters are not variables, you can look on them as such; in fact they are a form of local variable. They take the same form in the DEFinition as they do in the brackets after the name. So you might see:

```
DEF FNexample (a,b,c) = a*b*c
```

On the Spectrum and Acorn you can also have parameters which are strings; string functions are explained later in this article.

As the parameters are not ordinary variables, you can quite happily use variables and parameters with the same names, if you wish to, without fear of the computer harming the values of either.

HOW MANY PARAMETERS?

As with the names, the Commodores, Dragon and Tandy's DEF FN facility is less versatile than the other computers': it only allows one parameter. All other numbers used in the function must be actual numerical values that are present in the definition itself.

So, suppose you wanted to multiply a value by 9.81 as part of your function. You could use a parameter a for the value that is going to be changed by the function, and also use 9.81 in the definition.

The Spectrum and Acorn computers allow you to use as many parameters as you want. However, once you have set up a series of parameters in brackets in the DEF FN statement, you must include the same number of variables or numbers when you use the function. Even if you have allowed for more parameters than you actually want to use, you must still give values for them when you call the function. Type NEW and then this program to see this in practice (there is no Commodore, Dragon or Tandy program, since the point does not apply).

```
10 DEF FNa (a,b,c,d,e,f) = a*a*b
20 PRINT "please input two numbers"
30 INPUT a,b
40 PRINT FNa(a,b)
```

If you RUN this program, you will find that it does not work. But if you replace Line 4Ø with this:

```
40 PRINT FNa(a,b,Ø,Ø,Ø,Ø)
```

then suddenly it works, although you are not using any more information.

WHY USE A FUNCTION?

When is it better to use a function rather than just a direct instruction in a line?

The obvious advantage of using a defined function, rather than working out a set of calculations, is when you want to use the same calculation a large number of times. You can save both memory and time by having a user-defined function.

This is useful with long equations, or mathematical functions which you are going to use a lot in your programs. Here is an example, which rounds off a number (which you INPUT) to the number of decimal places which you specify, also by INPUTting a number. Since the function needs two parameters, it is only suitable for the Spectrum and Acorns.

```
1Ø FOR g=Ø TO 1 STEP Ø
2Ø PRINT ''' "Enter the number to round off,
   and the number of decimal places you
   want."'' "Please press ENTER after each..."
3Ø INPUT number, places
4Ø PRINT ''' number; "□to□"; places;
   "□places is", FN R(number, places)
5Ø PAUSE 1ØØ:NEXT g
6Ø DEF FN R(a,b)=INT (a*1Ø↑b+Ø.5)/1Ø↑b
```

```
1Ø REPEAT
2Ø PRINT''' "Enter the number to round off, and
   the□ □number of decimal places you
   want."'' "Please press RETURN after
   each..."
3Ø INPUT' NUMBER, PLACES
4Ø PRINT'''; NUMBER; "□to□"; PLACES;
   "□places is "; FNROUND(NUMBER,
   PLACES)
5Ø W=INKEY(2ØØ)
6Ø UNTIL Ø
7Ø DEF
FNROUND(A,B)=INT(A*1Ø∧B+Ø.5)/1Ø∧B
```

The DEFined FunctioN uses another function inside itself: the INT function, which converts







any number into an integer, or whole number. This shows an interesting point about user-defined functions: you can define a function to include results from other ROM, and user-defined, functions. But you cannot use a function in its own definition except with the Acorn computers. Whereas you can GOSUB to a subroutine from within itself, a function cannot call itself when it is being defined.

The Commodores, Dragon and Tandy, although they have only one parameter, can make good use of their user-defined function facility, too. Type in and RUN this program, which DEFines a FunctioN to work out the equation of an ellipse. You'll need Simons' Basic for the Commodore and the Super Expander Cartridge for the Vic.

When you RUN it, you will need to INPUT a few values, after which the program uses the function to plot two ellipses.

```
10 DEF FNY(X)=SQR(B*R*R-B*X*X/A)
40 PRINT "♥INPUT CONSTANTS
   A AND B": INPUT A,B
50 PRINT "INPUT THE TWO
   RADII":INPUT R1,R2
60 R=R1:Y1=FNY(Ø):R=R2:
   Y2=FNY(Ø)
70 IF Y2>Y1 THEN Y1=Y2
80 IF Y1>100 THEN R1=R1*100/Y1:
   R2=R2*100/Y1
90 HIRES Ø,1
100 FOR K=-159 TO 159
110 IF B*R1*R1<B*K*K/A THEN 130
120 R=R1:PLOT K+159,100
    +FNY(K),1:PLOT K+159,100-FNY(K),1
130 IF B*R2*R2<B*K*K/A THEN 150
140 R=R2:PLOT K+159,100
    +FNY(K),1:PLOT K+159,100-
    FNY(K),1
150 NEXT
160 GOTO 160
```

For the Vic 20, you should make a few changes to the Commodore version. First, replace the number 100 in Line 8Ø with 511. Then, replace the HIRES Ø,1 in Line 9Ø with GRAPHIC 2. The FOR ... NEXT loop in Line 10Ø should be FOR K=-511 TO 511, and not -159 TO 159. Again, in Line 12Ø and 14Ø the number 100 and the number 159 should be replaced by 511. On Lines 12Ø and 14Ø change PLOT to POINT, delete the last ,1 and insert 1, after POINT. With these changes, the program will RUN on the Vic.

```
10 DEF FNY(X)=SQR(B*R*R-
   B*X*X/A)
20 PMODE3,1
30 PCLS
40 CLS:INPUT" INPUT CONSTANTS A AND
   B□";A,B
50 INPUT" INPUT THE TWO RADII□";R1,R2
60 R=R1:Y1=FNY(Ø):R=R2:
   Y2=FNY(Ø)
70 IF Y2>Y1 THEN Y1=Y2
80 IF Y1>95 THEN R1=R1*95/Y1:
   R2=R2*95/Y1
90 SCREEN1,Ø
100 FOR K=-127 TO 128
110 IF B*R1*R1<B*K*K/A THEN 130
120 R=R1:PSET(K+127,95+
    FNY(K),2):PSET(K+127,96
    -FNY(K),2)
130 IF B*R2*R2<B*K*K/A THEN 150
140 R=R2:PSET(K+127,95+
    FNY(K),3):PSET(K+127,96
    -FNY(K),3)
150 NEXT
160 GOTO 160
```

The programs use the familiar Commodore, Dragon and Tandy graphics commands to draw the ellipses, using the function defined in Line 1Ø to work out the coordinates for each ellipse. As with the Spectrum and Acorn programs above, it uses another function in its definition: the computer's SQR function.

In case you may want to convert this program for the Spectrum or Acorn, here is the equation used to find the coordinates of the ellipse:

$$\frac{x^*x}{a}+\frac{y^*y}{b}=r^*r$$

In it, x is the x coordinate, and y the y coordinate, of each point on the circumference: r is the radius (in fact the program uses two values for r—R1 and R2, one for each ellipse). a and b are two constants, which you INPUT in the program.

STRING FUNCTIONS

As well as just numeric functions, the Acorn and Spectrum both also have string functions. You might be wondering what use a string function might have: type NEW, then this short program, and RUN it, to see.

```
10 PRINT "Please ENTER your name!"
20 INPUT n$
30 IF CODE n$>90 THEN LET n$=FN
u$(n$)
40 PRINT " "Hi, "; n$; "-aren't I clever?"
90 DEF FN u$(x$)=CHR$ (CODE
   x$-32)+x$(2 TO )
```

```
10 PRINT"PLEASE ENTER YOUR NAME"
20 INPUTN$
30 N$=FNLC(N$)
40 PRINT""HI, ";N$;"—AREN'T I CLEVER"
50 END
90 DEF FNLC(X$)
95 X$=CHR$(ASC(LEFT$(X$,1)) AND
   &5F)+RIGHT$(X$,LEN(X$)-1)
100 FOR T=2 TO LEN(X$):A$=MID$(X$,T,
    1): IF A$<"A" OR A$>"Z" THEN 120
110 X$=LEFT$(X$,T-1)+CHR$(ASC(A$)
    OR &20)+RIGHT$(X$,LEN(X$)-T)
120 NEXT
130 =X$
```

This program lets you INPUT a name, and then calls a user-defined string function to print out the first letter in capitals, the rest in lower case. (In fact the Spectrum INPUT is normally in lower case, so it capitalizes the first letter, while the Acorn version is normally in capitals, so it changes the rest of the word to lower case.)

User-defined string functions work in just the same way as numeric functions, except for the name of the function on the Spectrum. Instead of having a numeric variable name, they must have a string variable name. When you call the function, instead of using FN and a numeric name, you use FN and the string name.

The function can also have string parameters, which are used in just the same way as numeric parameters. You must of course put strings in the brackets after the FN when you call the function, if you do use string parameters.

To see this in practice, here is a routine which uses a defined string function to brighten up your INPUTed strings.

```
10 FOR f=0 TO 1 STEP 0
20 INPUT "INPUT your title□";i$
30 CLS
35 PRINT FN t$("□"+i$)
40 PAUSE 150
50 NEXT f
60 DEF FN t$(x$)=CHR$ 18+CHR$
   1+CHR$ 16+CHR$ 2+"******
   ******************************"
   +CHR$ 23+CHR$ (16-LEN x$/2)
   +x$+CHR$ 23+CHR$ 31+"******
   ******************************"
```

Note that since the Acorn version of this program uses MODE 7, it does not work properly on the Acorn Electron.

```
5 MODE 7
10 REPEAT
20 INPUT' "INPUT YOUR TITLE"
   'A$
30 CLS:PRINT FNTITLE(A$)
40 W=INKEY(300)
50 UNTIL 0
60 DEF FNTITLE(X$)=CHR$131
   +CHR$136+STRING$(36,
   "*")+"□□"+CHR$131+CHR$
   136+"*"+STRING$(17-LEN
   X$/2+(LEN X$ MOD 2),"□")
   +X$+STRING$(17-LEN X$/2,
   "□")+"*"+"□□"+CHR$131+
   CHR$136+STRING$(36,"*")
```

The DEF FN statement in Line 6Ø uses several CHR$s. These control the colour, and make the display flash to convert the strings, which you INPUT, into a pleasant title. The article on pages 314 to 320 explains how you can use these CHR$s. As with numeric functions, you can use the computer's built in string functions inside your definitions. The programs above use LEN to position the title, or name, exactly in the middle of the line.

Line 3Ø, which calls the function and then prints the new string at the top of the screen, also adds a space to the string.

The reason for this is so that even if you press [ENTER], or [RETURN], before typing any letters of a name (or, if you enter a 'null string') the computer still prints up something: the spaces.

OUT OF STEP

The strange-seeming STEP in Line 1Ø of the Spectrum program sends the computer into an infinite loop. It does this by telling the computer to count in STEPs of Ø, so that the computer never gets to the end of the loop. The Acorn does the same, but using REPEAT ... UNTIL Ø, instead.

While you could use a GOTO statement to do the same thing, GOTOs are usually considered as bad structuring in programs, so this is one way you can avoid having to use one.

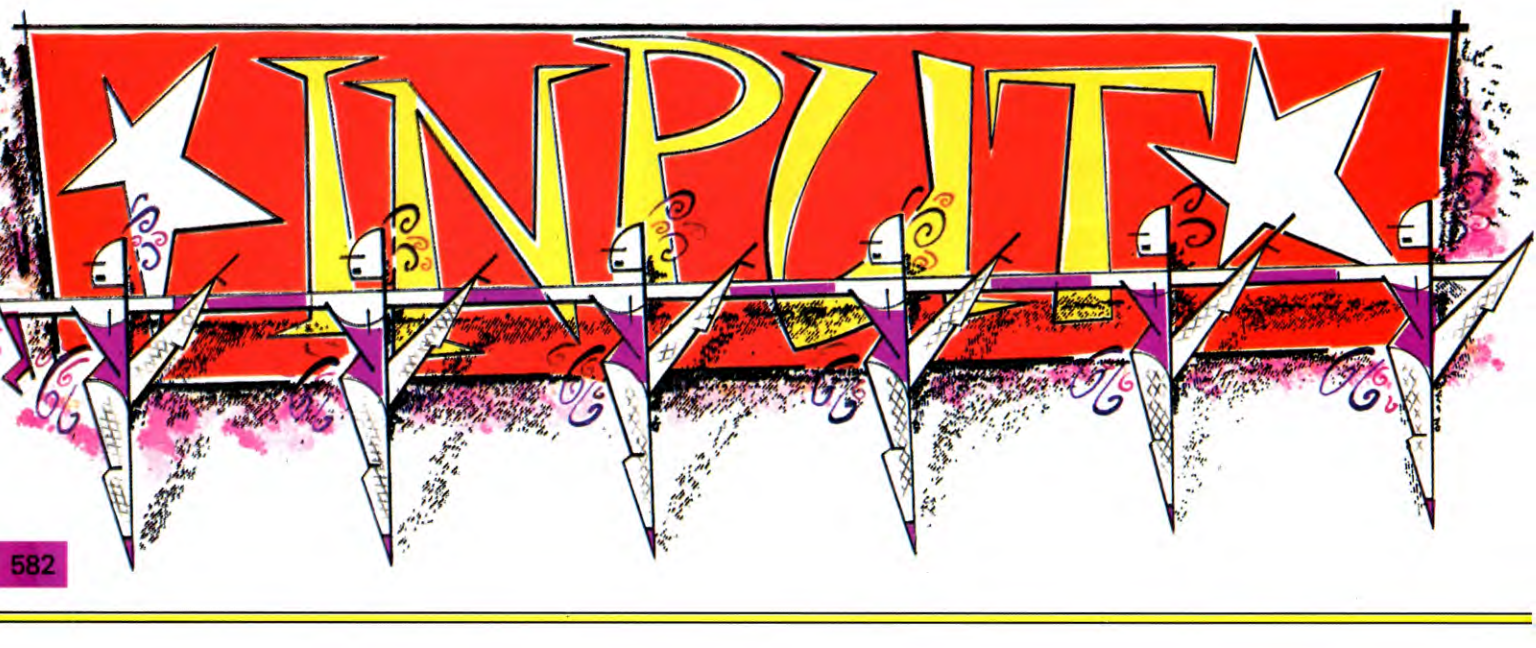
The programs in this article have all used DEFined FunctioNs which can be written in just one line of BASIC. This is because all except the Acorn computers are restricted to defining their functions in just one line. The Acorn computers, though, can take as many lines of BASIC as there is memory to define a function. The last line of the defining routine should begin '=' to tell the computer what the function actually equals.

The advantages with this are that you can have LOCAL variables, and change various factors before the function is actually defined. You could also include IF ... THEN conditions in the routine, which you can't with the other computers.

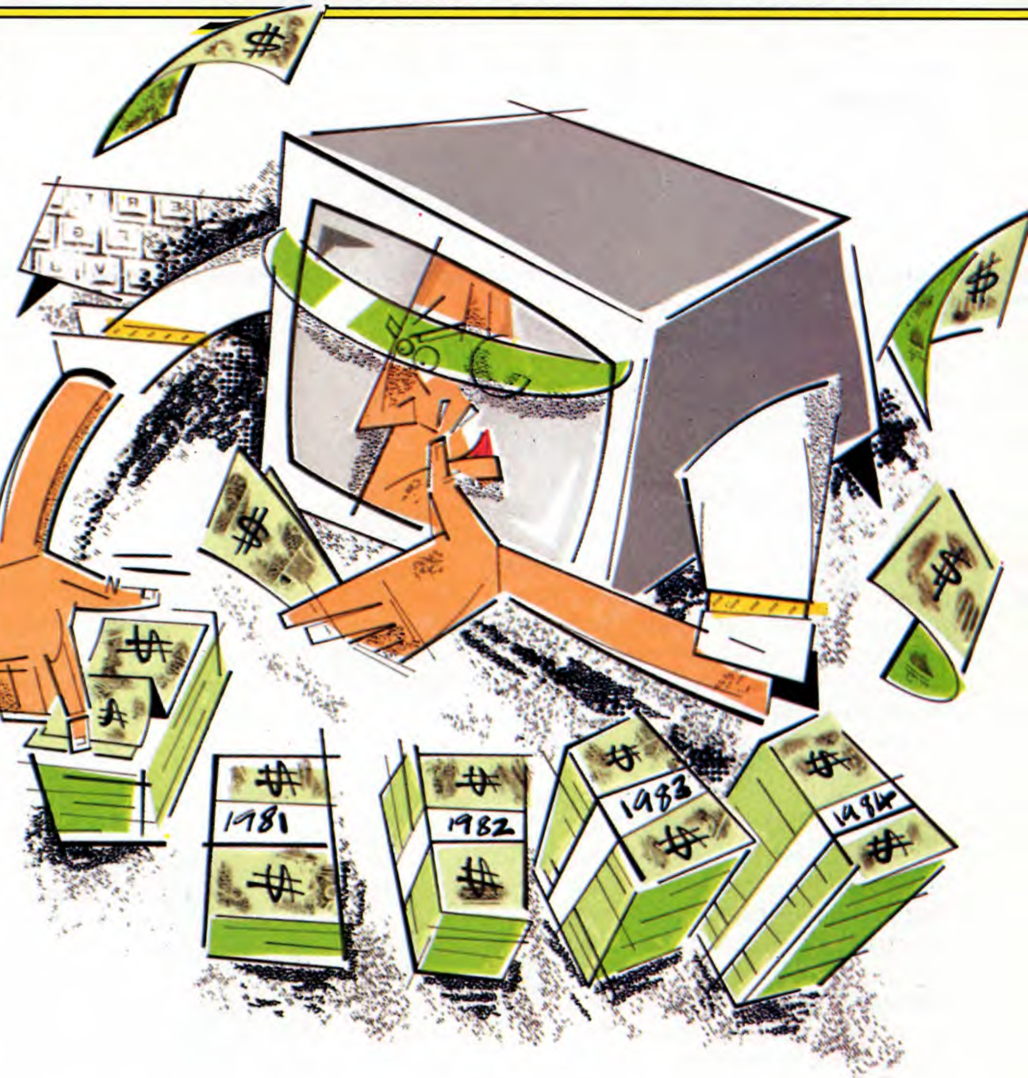
ADDING TO THE INTEREST

Here is another example of DEF FNs at work in the form of a utility program which, although short, can be very helpful if you are trying to work out how much interest you have earned on your savings.

The programs define a function to work



out the amount of money you should have after a given time period if you have been receiving a given rate of interest. You can INPUT the amount of money you have to start with, the rate of interest, and the time period.




```
10 DEF FN c(t)=INT (AM*((R/100
   +1)↑T)*100)/100
20 INPUT "How much (£)?□";AM
30 INPUT "Interest rate (%) ?□";R
40 INPUT "No. of time units?□";T
50 PRINT AT 9,0;"Total amount after
   interest= □";TAB 12;"";FN C(T)
60 PRINT INVERSE 1;AT 20,3;"Press
   any key to go again"
70 PAUSE 0: CLS : GOTO 20
```

```
10 CLS:PRINT''
20 INPUT"HOW MUCH",AM
30 INPUT"INTEREST RATE
   (%) ",R
40 INPUT"NUMBER OF TIME
   UNITS ",T
50 PRINT'''"TOTAL AMOUNT
   AFTER INTEREST=";FNC(T)
60 A$=INKEY$(200):PRINT'''
70 GOTO 20
100 DEF FNC(T)
110 =INT(AM*((R/100+
   1)↑T)*100)/100
```

```
10 DEF FN C(T)=INT(AM*((1+
   R/100)↑T)*100)/100
20 INPUT "HOW MUCH□";AM
30 INPUT "INTEREST RATE (%)□";R
40 INPUT "NO. OF TIME UNITS□";T
50 PRINT"TOTAL AMOUNT AFTER
   INTEREST =",FNC(T)
60 PRINT:PRINT"PRESS ANY KEY TO GO
   AGAIN"
70 A$=INKEY$:IF A$="" THEN 70
80 GOTO 20
```

For the Commodore and Vic, you should replace the A$=INKEY$ in Line 70 with GET A$. These programs work quite simply, by defining the function and then going through several INPUTs so you can give the details of your savings. The variables that the program uses are T—for the time period; AM—for the amount you have in the account to start off with; and R—the rate of interest.

The only parameter which the function uses is T. You can see this, since it is the only letter in brackets after the function's name when the function is being defined. You should note, though, that every time you use the function you can set three variables—only one of them is a parameter. This is useful, as it means that by mixing both variables and parameters, you can get around a limit on parameters.

Another point to note is that, with the Spectrum, although the function is defined in Line 10, the fact that it uses variables which have not yet been set up does not matter—the computer does not stop with an error.

The interest calculated is compound interest, so that you earn interest on the interest you earned in the last time period, etc.

This is automatically taken care of by the function. If, say, the interest rate is 10%, after one time period, you will have the original sum, plus 10/100 (=1/10 or .1) of the original sum. You can see this expression contained in the function, as R/100+1. So in the 10% example, the result after one time period is 1.1 times the amount you started with. After two time periods, it would be 1.1 times the new starting sum, which is already 1.1 times the original. So it is 1.1 times 1.1 times the amount you put in to start with. And 1.1*1.1 is 1.1↑T. This is the second part of the function—which takes the full form seen in the program, (R/100+1)↑T. The remaining part of the expression multiplies this by the original amount (AM) and rounds the result off to two significant places by multiplying by 100, using the INT function and dividing by 100 again.

The Acorn program is interesting, in that it takes more than one program line to define its function.

You might like to use the two formulae below to create user-DEFined FunctioNs which your computer does not have. The formulae are for ARC SIN and ARC COS, for the Commodores, Dragon and Tandy:

```
ASN(X)=ATN (X/SQR(-X*X+1)
ACS(X)=-ATN
   (X/SQR(-X*X+1))+1.5708
```

You can also use formulae for other mathematical functions.

BOUNCING AROUND IDEAS

Mathematics provides many models for the real world—models which you can use in your programs as the basis for effective displays. In this example, a graphic bouncing ball . . .

Applied mathematics, and other branches of science, make use of a large number of equations in an attempt to provide a model for the real world. Such equations are derived as the result of observation and experimentation. The data that these generate is examined, and an equation that explains the relationship is derived to suit the data.

Such equations have been worked out for all sorts of natural and physical phenomena—everything from the growth of a plant to the path traced out by a comet moving through space. There are all kinds of uses for this information, since by working through the equations it is possible to predict how the real thing will behave.

The aim of some of the more obscure equations may not always appear immediately obvious—unless you happen to be involved in a particularly abstruse branch of research. But many of them provide models of things which are with in everyone's experience. You have already seen one such example used in a computer program, on page 411, where it was used to predict how a falling object would behave.

But the result of this calculation is simply another number—for time or whatever—which is just another model for the real world. This can often be very useful in itself, but there is another way in which such equations can be programmed into your computer which provides an even more vivid model and which has almost limitless applications. This is by using the equation to control a screen display directly. So in the earlier example, you could have a moving graphic to show the path of the falling object.

For that matter, you could equally well turn any equation of motion into something which would manipulate the screen image. The ability to do this is the basis of many games programs and dramatic animations.

The best way to see how this can be done is to work through an actual example—taking a series of simple equations which could be found in any physics text book and turning them into a vivid screen demonstration. For this example, let's start with something which is very easy to model—the path of a ball on a pool table after it is struck by the cue.

MODELLING THE BALL

If you strike a ball with a cue (or kick it for that matter), it will move in a straight line until it hits something, or unless some other force is acting on it—like gravity or a cross-wind, say. On a smooth pool table, you can effectively ignore the chances of cross-winds—so there is really only one force to worry about. This is the effect of friction, which will slow the ball down gradually as it rolls. But by how much? This is where we need the first of our equations.

Roughly speaking, the harder the ball is hit (the larger the force given to it), the further it travels. There is an equation which works out how far the object travels, given the initial force, and certain other facts such as the effect of friction. This equation has a number of interesting applications in BASIC programming. To see it at work type in and RUN this program.



```
10 CLS
20 PRINT "1) Football in long grass"
30 PRINT '"2) Golfball in short grass"
40 PRINT '"3) Golfball on the moon"
50 PRINT '"4) Poolball on a pool table"
60 INPUT "which option would you like?",opt
70 IF opt<1 OR opt>4 THEN
   GOTO 60
80 IF opt=1 THEN LET fg=49
90 IF opt=2 THEN LET fg=34.5
95 IF opt=3 THEN LET fg=1.64
100 IF opt=4 THEN LET fg=17.7
110 CLS
115 PRINT "The speed in metres per
   second□ □can be between 1 and 15"
120 INPUT "What initial speed do you
   want□ □to give it? (mps)",v
125 IF v<1 OR v>15 THEN GOTO 120
160 LET dist=(v*v)/(2*fg)
170 CLS
210 PRINT AT 2,4;"your ball would travel"
220 PRINT AT 7,8;dist;"□metres"
250 PRINT AT 18,3; FLASH 1;"press any key to
   go again"
260 PAUSE 0
270 GOTO 10
```



```
10 PRINT "♥"
20 PRINT "1) FOOTBALL IN LONG GRASS"
30 PRINT "2) FOOTBALL IN SHORT GRASS"
40 PRINT "3) GOLFBALL ON THE MOON"
50 PRINT "4) POOLBALL ON A POOL TABLE"
60 PRINT "■ENTER OPTION ?"
70 GET K$:IF K$<"1" OR K$>"4" THEN
   70
80 OP=VAL(K$)
90 IF OP=1 THEN FG=4
100 IF OP=2 THEN FG=1.4
110 IF OP=3 THEN FG=.1
120 IF OP=4 THEN FG=.7
130 PRINT "♥"
140 PRINT "THE SPEED IN METRES PER
   SECOND CAN BE"
145 PRINT "BETWEEN 1 AND 15."
150 PRINT "WHAT INITIAL SPEED DO YOU
   WANT TO GIVE IT (MPS)."
155 INPUT VE
160 IF VE<1 OR VE>15 THEN 130
170 DI=(VE*VE)/(2*FG)
180 PRINT "♥"
190 PRINT "YOUR BALL WOULD
   TRAVEL";DI;"METRES"
200 PRINT "■■HIT ANY KEY"
210 GET K$:IF K$="" THEN 210
220 RUN
```

```
5 @%=&20204
10 MODE1
20 PRINT'''"1) FOOTBALL IN LONG GRASS"
30 PRINT''"2) GOLFBALL IN SHORT GRASS"
40 PRINT''"3) GOLFBALL ON THE MOON"
50 PRINT''"4) POOLBALL ON A POOL TABLE"
60 PRINT'''"WHICH OPTION DO YOU
   WANT ?"
70 G=GET-48:IF G<1 OR G>4 THEN 70
80 IF G=1 THEN A=4
90 IF G=2 THEN A=1.4
100 IF G=3 THEN A=.3
110 IF G=4 THEN A=.7
120 CLS
130 PRINT'''"THE RANGE OF VELOCITIES
   CAN BE:-
   FROM 1 TO 15"
140 INPUT"WHAT INITIAL VALUE WOULD
   YOU LIKE IN MPS",V
```

```
150 IF V<1 OR V>15 THEN 140
160 S=V*V/(2*A)
170 CLS
180 PRINT''"YOUR BALL WOULD
    TRAVEL ";S;" METRES"
190 PRINT''''"PRESS ANY KEY TO REPEAT"
200 G=GET:GOTO 10
```

```
10 CLS
20 PRINT@35,"1) FOOTBALL IN LONG
   GRASS"
30 PRINT@67,"2) GOLFBALL IN SHORT
   GRASS"
40 PRINT@99,"3) GOLFBALL ON THE
   MOON"
50 PRINT@131,"4) POOLBALL ON A POOL
   TABLE"
60 PRINT:PRINT"□WHICH
   OPTION WOULD
   YOU LIKE ?"
70 K$=INKEY$:IFK$<"1"ORK$>"4"
   THEN70
80 OP=VAL (K$)
90 IF OP=1 THEN FG=4
100 IF OP=2 THEN FG=1.4
110 IF OP=3 THEN FG=.1
120 IF OP=4 THEN FG=.7
130 CLS
140 PRINT"□THE SPEED IN METRES PER
    SECOND CAN BE BETWEEN 1 AND 15"
150 PRINT:INPUT"□WHAT INITIAL SPEED
    DO YOU WANT TO GIVE IT (MPS) ";VE
160 IF VE<1 OR VE>15 THEN 130
170 DI=(VE*VE)/(2*FG)
180 CLS
190 PRINT"□YOUR BALL WOULD
    TRAVEL ",DI;"METRES"
200 PRINT@449,"PRESS ANY KEY TO GO
    AGAIN"
210 IF INKEY$="" THEN210
220 GOTO 10
```

The program begins by asking you about the conditions in which the ball is travelling. It then calculates the distance that the object would travel, for any choice of initial speed, which you INPUT. It takes into account a number of factors, such as friction, mass, and gravity. The equation it uses looks like this:

$$\text{Distance} = \frac{v\uparrow 2}{2*fg}$$

where v is the speed, which you INPUT, and fg is a value comprising gravity and friction Obviously, the value



of fg is different for each of the options in the program.

WORKING OUT THE ANSWER

The equation is solved (worked out), in Line 16Ø on the Spectrum and Acorn or Line 17Ø on the other machines. You can see that although the equation contains a power there is not a power sign in the program line. This is becuase the computers work out multiplication sums much faster than they work out powers. So when the number is just squared (as here) it is better to multiply *v* by itself than to 'raise it to the power of 2'.

Option 3, the golf ball travelling on the moon, gives so much larger results than the other because its value of fg is far smaller than those of the other options. The reason for this is that gravity on the moon is just one sixth the size it is on Earth.

The last option, the pool ball on a pool table, also gives somewhat larger results than the other two options. This is because friction on a pool table is much less than it is in long or short grass; another object which would travel similarly well is a puck in an ice rink, where friction is similarly low.

The values in the program for friction and gravity are only estimated, and are not accurate, but give a reasonable idea of how an equation like this can be made into a working program to tell you how far an object might travel under the various circumstances. But we still do not have a moving object.

A BOUNCING BALL

The equation above calculates how far an object travels, but it assumes that there is nothing in the way of the object.

Using the pool ball option in the program above is a good example: it is all very well for the ball to travel 1Ø metres, or whatever answer you might get, but pool tables are not that long: the ball would bounce off the sides of the table.

Unlike a football bouncing on a grass field, it is quite simple to work out how the pool ball reacts when it hits the cushion (unless given a crafty spin). The ball simply bounces off mirroring the 'angle of incidence' (the angle at which the ball hits the side to start with).

If you look at **fig. 1** you can see this very clearly. The ball was hit from point a, and moved along the path shown by the dotted line, until it stopped at point b. As you can see, the angles are symmetrical.

Once again, the world of applied mathematics provides a heady equation to demonstrate this principle. Armed with this, you can soon have a program which shows a ball/puck bouncing off any, or all, of the sides.

The panel on page 592 shows the equations to work out the various different positions of the ball after bouncing off any of the sides. While these look frightening on paper, your computer can use them easily, and very quickly.

The Commodore 64 version of the following program is written in Simons' BASIC and needs a Simons' cartridge, while the Vic 20 needs a Super Expander.

```
10 BORDER 4: PAPER 4: INK Ø: CLS
60 LET bx=128: LET by=76: LET cx=100:
   LET cy=112: LET nx=100: LET ny=112
90 FOR n=6 TO 20: PRINT PAPER
   7;AT n,2;"□□□□□□□□□□
   □□□□□□□□□□□□□□□□□
   □□□□": NEXT n
100 CIRCLE OVER 1;bx,by,4
110 PLOT OVER 1;cx-4,cy: DRAW OVER
   1;8,Ø: PLOT OVER 1;cx,cy-4: DRAW
   OVER 1;Ø,8
120 LET a$=INKEY$: IF a$="" THEN GOTO
   120
130 IF INKEY$="a" AND cy>12 THEN LET
   ny=cy-2: GOTO 190
140 IF INKEY$="q" AND cy<122 THEN LET
   ny=cy+2: GOTO 190
150 IF INKEY$="p" AND cx<236 THEN LET
   nx=cx+2: GOTO 190
160 IF INKEY$="o" AND cx>20 THEN LET
   nx=cx-2: GOTO 190
170 IF INKEY$=CHR$ 32 THEN GOSUB 500:
   GOTO 300
190 PLOT OVER 1;cx-4,cy: DRAW OVER
   1;8,Ø: PLOT OVER 1;cx,cy-4: DRAW
   OVER 1;Ø,8
200 LET cy=ny: LET cx=nx
210 GOTO 110
300 IF INKEY$="" THEN GOTO 300
305 CLS
310 PRINT AT 2,5;"DISTANCE
   MOVED= ";INT (p*p*100/(18*18*.4))/100
320 PRINT AT 5,5;"INITIAL
   VELOCITY= ";INT (ABS (p*100/18))/100
330 PRINT AT 10,8;"ANOTHER GO (Y/N)?"
340 LET a$=INKEY$: IF a$<>"y" AND
   a$<>"n" THEN GOTO 340
350 IF a$="y" THEN CLS : GOTO 90
360 PAPER 7: CLS : STOP
500 IF cx=bx AND cy=by THEN LET p=Ø:
   RETURN
510 LET p=Ø
515 PRINT AT Ø,Ø;"□□□□□□□□□
   □□□□□□□□□□□□□□□□
   □□□□□□□□□□□"
520 LET p=p+2: IF p=256 THEN GOTO
   510
530 PLOT p-1,168: DRAW Ø,7: PLOT p,168:
   DRAW Ø,7
```

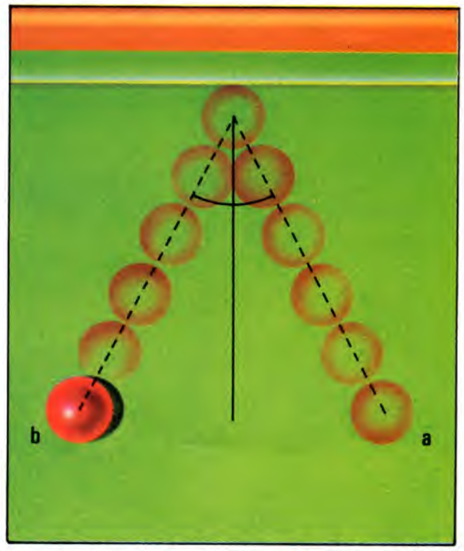


Fig. 1. A snooker ball's path after hitting the table's side

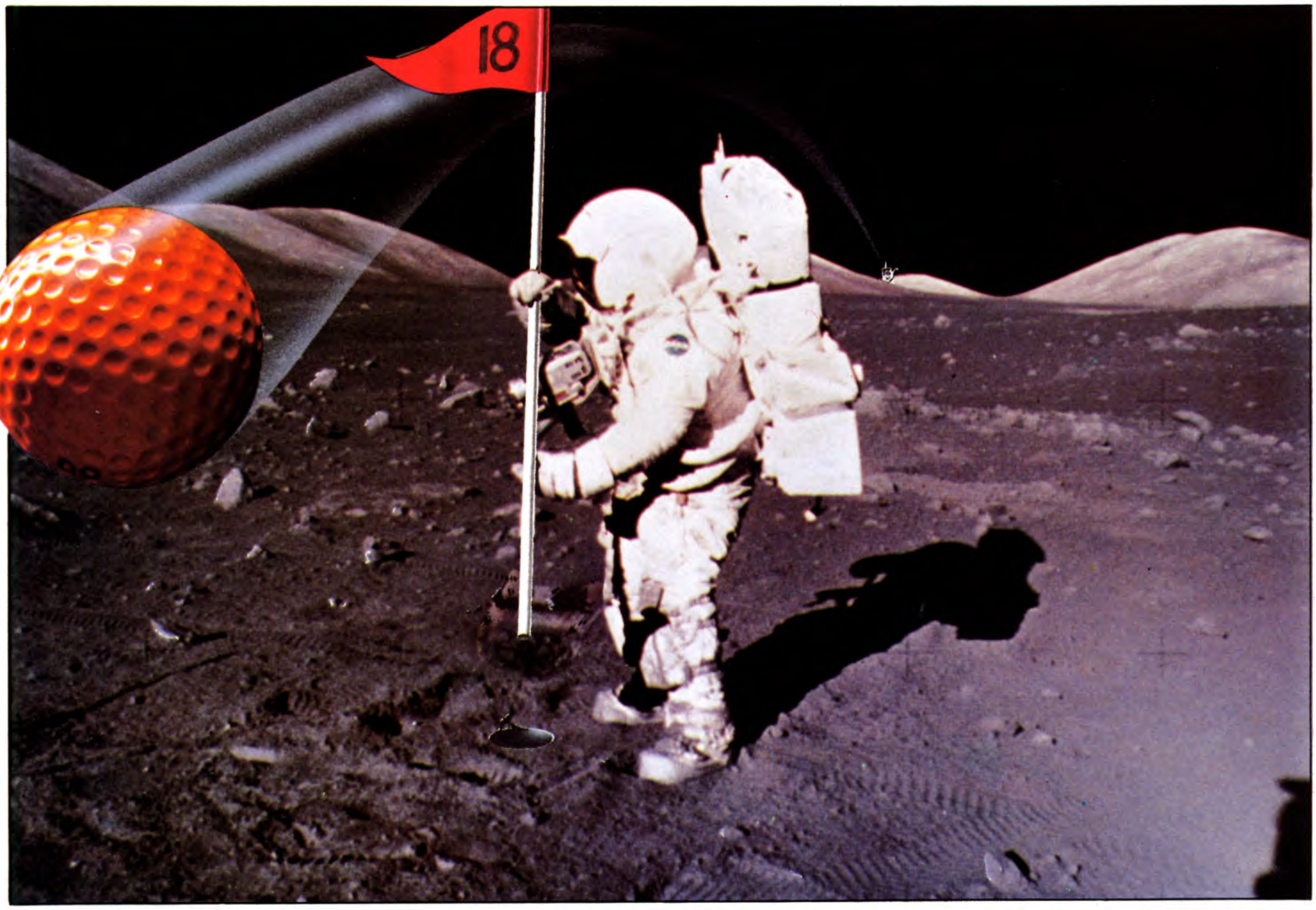
```
540 IF INKEY$=CHR$ 32 THEN GOTO 520
550 CIRCLE OVER 1;bx,by,4
560 LET dx=cx-bx: LET dy=cy-by
570 LET v=p/18: LET sq=SQR
   (dx*dx+dy*dy)
590 LET t=1
600 PLOT bx+.5,by+.5
610 LET vx=v*dx/sq: LET vy=v*dy/sq
620 LET bx=bx+vx*t-SGN vx*.1*t*t: LET
   by=by+vy*t-SGN vy*.1*t*t
630 LET v=v-.1
640 IF ABS v<.1 THEN GOTO 700
650 IF bx<15 THEN LET dx=-dx: LET
   bx=30-bx
660 IF bx>239 THEN LET dx=-dx: LET
   bx=478-bx
670 IF by<8 THEN LET dy=-dy: LET
   by=16-by
680 IF by>127 THEN LET dy=-dy: LET
   by=254-by
690 GOTO 600
710 IF bx<19 THEN LET bx=20
720 IF bx>235 THEN LET bx=235
730 IF by<12 THEN LET by=12
740 IF by>123 THEN LET by=123
750 CIRCLE bx,by,4: RETURN
```

```
10 HIRES Ø,1:MULTI 6,1,5:
   COLOUR Ø,Ø
20 BX=80:BY=100:CX=BX:
   CY=BY
30 BLOCK Ø,30,160,180,1
35 BLOCK 10,40,150,170,3
40 TEXT Ø,13,"POWER",3,1,5
45 TEXT BX,BY,"○",Ø,1,1
50 FOR Z=Ø TO 3:LINE 1,Z,160,Z,1:
   NEXT Z
60 GET A$:IF A$="" THEN 110
```

```
70 IF A$="▮▮" AND CX>14 THEN
   CX=CX-3
80 IF A$="▮▯" AND CX<138 THEN
   CX=CX+3
90 IF A$="▢" AND CY>44 THEN
   CY=CY-3
100 IF A$="▣" AND CY<156 THEN
   CY=CY+3
110 FOR N=1 TO 2
120 TEXT CX,CY,"+",4,1,1:NEXT N
130 IF A$<>"□" OR(CX=BX AND
   CY=BY)THEN 6Ø
135 P=Ø
140 P=P+2
150 PLOT P-1,2,2:PLOT P,2,2:IF P=16Ø
   THEN LINE 1,2,161,2,1:GOTO 135
160 IF PEEK(197)=6Ø THEN 14Ø
170 TEXT BX,BY,"◙",3,1,1
180 DX=CX-BX:DY=CY-BY
190 V=P/1Ø:SQ=SQR(DX*DX+DY*DY)
200 LX=BX:LY=BY
210 T=1
220 TEXT LX+.5,LY+.5,"◙",4,1,1:
   BX=LX+.5:BY=LY+.5
230 VX=V*DX/SQ:VY=V*DY/SQ
240 LX=LX+VX*T-SGN(VX)*.1*T*T:
   LY=LY+VY*T-SGN(VY)*.1*T*T
250 V=V-.1
255 IF ABS(V)<.1 THEN BX=LX:
   BY=LY:GOTO 32Ø
260 TEXT BX,BY,"◙",4,1,1
270 IF LX<1Ø THEN DX=-DX:
   LX=2Ø-LX
280 IF LX>143 THEN DX=-DX:
   LX=286-LX
290 IF LY<4Ø THEN DY=-DY:
   LY=8Ø-LY
300 IF LY>162 THEN DY=-DY:
   LY=324-LY
305 PLOT BX+3,BY+3,1
310 GOTO 22Ø
320 BLOCK Ø,183,16Ø,2ØØ,1
330 TEXT Ø,184,"□□DISTANCE
   MOVED□:",2,1,5
340 TEXT Ø,192,"□INITIAL VELOCITY:",2,1,5
350 TEXT 9Ø,184,STR$(INT(P*P*1ØØ/
   (18*18*.4))/1ØØ),3,1,6
360 TEXT 9Ø,192,STR$(INT(ABS(P*1ØØ/
   18))/1ØØ),3,1,6
370 TEXT 1Ø,22,"ANOTHER GO (Y/N)",2,1,8
380 GET A$:IF A$="N" THEN PRINT
   "▢▤":NRM:END
390 IF A$<>"Y" THEN 38Ø
400 TEXT 1Ø,22,"ANOTHER GO (Y/N)?",
   Ø,1,8
410 GOTO35
```

```
10 GRAPHIC 1
20 BX=512:BY=54Ø:CX=BX:CY=BY
30 COLOR 5,6,2,Ø:DRAW 1,Ø,95 TO 1Ø23,95
35 PAINT 3,Ø,Ø
40 CHAR Ø,Ø,"POWER"
45 CIRCLE 1,BX,BY,24,24
60 GET A$:IF A$="" THEN 11Ø
70 IF A$="▮▮" AND CX>2Ø THEN
   CX=CX-8
80 IF A$="▮▯" AND CX<1ØØØ THEN
   CX=CX+8
90 IF A$="▢" AND CY>12Ø THEN
   CY=CY-8
100 IF A$="▣" AND CY<1ØØØ THEN
   CY=CY+8
110 FOR N=3 TO Ø STEP-1
120 POINT N,CX,CY:NEXT N
130 IF A$<>"□" OR(CX=BX AND
   CY=BY)THEN 6Ø
135 P=Ø
140 P=P+2
150 DRAW 2,Ø,7Ø TO P*1Ø,7Ø:IF P>1ØØ
   THEN:DRAW 3,Ø,7Ø TO 1Ø23,7Ø:P=1
160 IF PEEK(197)=32 THEN 14Ø
170 CIRCLE Ø,BX,BY,24,24
180 DX=CX-BX:DY=CY-BY
190 V=P/2.7:SQ=SQR(DX*DX+DY*DY)
210 T=1
220 BX=BX+.5:BY=BY+.5
230 VX=V*DX/SQ:VY=V*DY/SQ
240 BX=BX+VX*T-SGN(VX)*.1*T*T:
   BY=BY+VY*T-SGN(VY)*.1*T*T
250 V=V-.1
255 IF ABS(V)<.1 THEN 32Ø
280 IF BX>1Ø23 THEN DX=-DX:
   BX=2Ø46-BX
290 IF BY<1ØØ THEN DY=-DY:
   BY=2ØØ-BY
300 IF BY>1Ø23 THEN DY=-DY:
   BY=2Ø46-BY
305 IF BX<Ø THEN DX=-DX:
   BX=Ø-BX
306 POINT 2,BX,BY
310 GOTO 22Ø
320 IF BX<24 THEN BX=24
321 IF BX>999 THEN BX=999
322 IF BY<124 THEN BY=124
323 IF BY>999 THEN BY=999
324 CIRCLE 1,BX,BY,24,24
325 POKE 198,Ø:WAIT 198,1:
   POKE 198,Ø:GRAPHIC Ø:
   REGION Ø
330 PRINT "DISTANCE MOVED :"
340 PRINT INT(P*P*1ØØ/(18*
   18*.4))/1ØØ
350 PRINT "▣INITIAL VELOCITY:"
360 PRINT INT(ABS(P*1ØØ/18))/1ØØ
370 PRINT "▣▣▮▯▮▯ANOTHER GO
   (Y/N)?"
380 GET A$:IF A$="N" THEN PRINT
   "▢▤":END
390 IF A$<>"Y" THEN 38Ø
400 GRAPHIC 1
410 GOTO 3Ø
```

```
10 MODE1
20 VDU 5
30 VDU 23,224,Ø,24,6Ø,126,126,6Ø,24,Ø
40 PROCBLOCK(15Ø,5Ø,2ØØ,65Ø):
   PROCBLOCK(15Ø,6ØØ,115Ø,65Ø)
50 PROCBLOCK(15Ø,5Ø,115Ø,1ØØ):
   PROCBLOCK(11ØØ,5Ø,115Ø,65Ø)
60 VDU19,Ø,2,Ø,Ø,Ø
70 MOVE 5Ø,732:PRINT"POWER"
80 BX=65Ø:BY=35Ø:CX=65Ø:
   CY=35Ø
90 GCOL3,3
100 MOVE BX-12,BY-12:VDU 224
110 MOVE CX-12,CY-12:VDU 43
120 NX=CX:NY=CY
130 IF INKEY(-87) THEN NY=NY-8
140 IF INKEY(-56) THEN NY=NY+8
150 IF INKEY(-98) THEN NX=NX-8
160 IF INKEY(-67) THEN NX=NX+8
170 IF INKEY(-99) AND NOT(NX=BX AND
   NY=BY) THEN PROCSHOT:GOTO9Ø
180 IF NX=CX AND NY=CY THEN 12Ø
190 IF NX<212 OR NX>1Ø86 OR NY<112
   OR NY>596 THEN 12Ø
200 MOVE CX-12,CY+12:VDU 43
210 CX=NX:CY=NY
220 MOVE CX-12,CY+12:VDU 43
230 GOTO 12Ø
240 DEF PROCSHOT
250 VDU 4:PRINTTAB(Ø,3)STRING$
   (16Ø,"□")
260 GCOLØ,Ø:PROCBLOCK(3ØØ,7ØØ,
   1158,732):GCOLØ,3
270 P=1Ø
280 IF P>85Ø THEN P=1Ø:GCOLØ,Ø:
   PROCBLOCK(3ØØ,7ØØ,1158,732):
   GCOLØ,3
290 PROCBLOCK(3ØØ+P,7ØØ,
   3ØØ+P+8,732)
300 SOUND1,-15,P*255/85Ø,1
310 IF INKEY(-99) THEN P=P+8:
   GOTO 28Ø
320 VDU 24,2Ø4;1Ø4;1Ø96;596;:
   CLG:VDU26
330 DX=CX-BX:DY=CY-BY
340 V=P/2Ø:SQ=SQR(DX*DX+DY*DY)
360 T=1.5
370 PLOT69,BX,BY
380 VX=V*DX/SQ:VY=V*DY/SQ
390 BX=BX+VX*T-SGN(VX)*.1*T*T:
   BY=BY+VY*T-.1*T*T
400 V=V-.1
410 IF ABS(V)<.1 THEN 48Ø
420 IF BX<2ØØ THEN DX=-DX:
   BX=4ØØ-BX:SOUND1,-15,1ØØ,1
430 IF BX>11ØØ THEN DX=-DX:
   BX=22ØØ-BX:SOUND1,-15,1ØØ,1
440 IF BY<1ØØ THEN DY=-DY:
   BY=2ØØ-BY:SOUND1,-15,1ØØ,1
```



```
450 IF BY>6ØØ THEN DY=−DY:
    BY=12ØØ−BY:SOUND1,−15,1ØØ,1
460 GOTO 37Ø
480 IF BX<212 THEN BX=212
490 IF BX>1Ø8Ø THEN BX=1Ø8Ø
500 IF BY<12Ø THEN BY=12Ø
510 IF BY>588 THEN BY=588
520 VDU 4:PRINTTAB(Ø,3)“INITIAL
    VELOCITY□”,P/2Ø
530 PRINTTAB(Ø,6)“DISTANCE
    TRAVELLED□”,(P/2Ø)∧2/.5:VDU 5
540 ENDPROC
550 DEF PROCBLOCK(X,Y,X2,Y2)
560 MOVE X,Y:MOVEX2,Y
570 PLOT85,X,Y2:PLOT85,X2,Y2
580 ENDPROC
```

On the Tandy, change the 223 in Lines 13Ø, 14Ø, 15Ø, 16Ø, 17Ø, and 54Ø to 247.

```
10 PMODE3,1:PCLS:DIM B(2),BL(2),C(Ø)
30 DRAW“BM5,ØC3FRFD6GLGHL2HU6E”:
    PAINT(5,5),3
40 DRAW“BM2Ø,ØC2D2NLR2DL3FD”
50 GET(Ø,Ø)−(9,1Ø),B,G
60 GET(18,Ø)−(23,5),C,G
70 PCLS4:SCREEN1,Ø
80 LINE(18,58)−(23Ø,17Ø),PRESET,BF
90 BX=123:BY=11Ø:CX=125:CY=112:
    NX=125:NY=112
100 PUT(BX,BY)−(BX+9,BY+1Ø),B,OR
110 PUT(CX,CY)−(CX+5,CY+5),C,OR
120 NX=CX:NY=CY
130 IF PEEK(341)=223 THEN NY=NY−2:
    GOTO 19Ø
140 IF PEEK(342)=223 THEN NY=NY+2:
    GOTO 19Ø
150 IF PEEK(343)=223 THEN NX=NX−2:
    GOTO 19Ø
160 IF PEEK(344)=223 THEN NX=NX+2:
    GOTO 19Ø
170 IF PEEK(345)=223 GOSUB 5ØØ:
    GOTO 22Ø
180 GOTO 1ØØ
190 IF NX<18 OR NX>225 OR NY<58 OR
    NY>165 THEN 12Ø
200 PUT(CX,CY)−(CX+5,CY+5),BL,PSET
210 CX=NX:CY=NY:GOTO 1ØØ
220 PUT(BX,BY)−(BX+9,BY+1Ø),B,OR:
    PUT(CX,CY)−(CX+5,CY+5),C,OR
230 A$=INKEY$
240 IF INKEY$=“” THEN 24Ø
250 CLS:PRINT@33,“DISTANCE MOVED=”;
    INT(P*P*1ØØ/(18*18*.4))/1ØØ
260 PRINT@97,“INITIAL VELOCITY =”;
    INT(ABS(P*1ØØ/18))/1ØØ
270 PRINT@225,“ANOTHER GO (Y/N) ?”
280 A$=INKEY$:IF A$<>“Y” AND
    A$<>“N” THEN 28Ø
290 IF A$=“Y” THEN SCREEN1,Ø:GOTO 1ØØ
300 CLS:END
500 IF CX=BX+2 AND CY=BY+2 THEN
    RETURN
510 P=1
520 P=255AND(P+2)
530 LINE(P,Ø)−(P+2,6),PSET,BF:
    LINE(P+4,Ø)−(255,6),PRESET,BF
540 SOUNDP,1:IF PEEK(345)=223 THEN52Ø
550 LINE(18,58)−(23Ø,17Ø),PRESET,BF
560 DX=CX−BX−2:DY=CY−3−BY
570 V=P/18:SQ=SQR(DX*DX+DY*DY)
580 BX=BX+4:BY=BY+5:T=1
600 PSET(BX+.5,BY+.5,3)
610 VX=V*DX/SQ:VY=V*DY/SQ
620 BX=BX+VX*T−SGN(VX)*.1*T*T:
    BY=BY+VY*T−SGN(VY)*.1*T*T
630 V=V−.1: IF ABS(V)<.1 THEN 7ØØ
650 IF BX<18 THENDX=−DX:
    BX=36−BX:SOUND5,1
660 IF BX>23Ø THENDX=−DX:
```

```
    BX=460-BX:SOUND5,1
670 IF BY<58 THEN DY=-DY:
    BY=116-BY:SOUND5,1
680 IF BY>170 THEN DY=-DY:
    BY=340-BY:SOUND5,1
690 GOTO 600
700 BX=BX-4:BY=BY-5
710 IF BX<18 THEN BX=18
720 IF BX>221 THEN BX=221
730 IF BY<58 THEN BY=58
740 IF BY>160 THEN BY=160
750 RETURN
```

When you RUN this program, the computer draws a rectangle in the middle of the screen: this is the rink, table or the area in which the object can move about.

A MOVING TARGET

You can also see a ball in the middle of the rectangle. In fact, there is also a cursor which may be concealed by this ball. You can see this by moving the cursor with the following keys: Spectrum up Q; down A; left O; right P; Commodore and Vic 20 up ↑; down ↓; left ←; right →; (cursor keys) Acorn up P; down L; left Z; right X; Dragon and Tandy up↑; down ↓; left ←; right →; (cursor keys). On all except the Vic, the cursor is a cross. The Vic's is a dot, to speed up the program. You can move the cursor where you like within the rectangle to indicate the direction of movement of the ball. When you have positioned the cursor where you want it, press the space bar (or the SPACE key on the Spectrum).

Then a horizontal line appears at the top of the screen, and gets steadily larger as long as you hold the space bar or key down. This shows how hard you will hit the ball—the longer the line the greater the force.

As soon as you release the space bar or key, the computer clears the rectangle, and then plots a series of dots: these show the path that the ball would take. The dots are much closer together when the ball is slowing down.

Try putting the cursor in different positions and giving different velocities to the ball to see the effect.

HOW IT WORKS

You can see the principles of reflection very clearly here. Whenever the ball hits a side, it bounces back, always at the same angle it came in at. This is the same for every side of the table. This is arranged in slightly different ways on each computer.

The programs all start by initialising some variables and setting the correct screen mode and colours. The Dragon and Tandy program also DIMensions 3 arrays, and GETs two UDGs into them (one for the ball, one for the cross and one for a blank).

The variables set up are BX, BY (x and y coordinates for the ball); CX, CY (the coordinates for the cross); and NX and NY (the 'new coordinates for the cross—these are the variables used and changed when you move the cursor around in the rectangle).

Once this is done, the rectangle is drawn on the screen, and the INKEY$, or GET$, routine which lets you move the cursor comes into effect. The Dragon/Tandy routine PEEKs the keyboard instead of using INKEY$, since this allows all keys to auto-repeat.

Apart from just checking to see whether you are moving the cross up, down, left, or right, the routine also checks for the space bar, or key, being pressed. If you are pressing the space key, the computer GOSUBs to 500 (except the Commodores, which do not use subroutines; the Commodores just carry on to the next Lines if the space is pressed). This routine first checks on the position of the cross or dot; if it is over the centre of the ball, then the power is immediately set to 0, and the computer RETURNs from the subroutine. The Dragon does not specifically set P (P is the variable for the power) to 0, as its BASIC automatically assumes every variable to be 0 unless you tell is otherwise.

The reason for this is that when the cross is on top of the ball, the computer does not know in which direction to move the ball—so this check prevents an error occurring.

If the cross is in an allowed position, then the power factor, P, is set to 0 on the Spectrum, or 1 on the Commodores, Dragon, and Tandy. You can see this, as the horizontal line is wiped out at this point.

INCREASING THE POWER

P is then increased by 2 every time that the computer detects the space key being pressed, until it reaches 256 (255 on the Dragon and Tandy). Once this happens, P is reset to 0—and the horizontal line, which increases with P, is removed again. The computer stays in this loop until the space key is no longer being pressed.

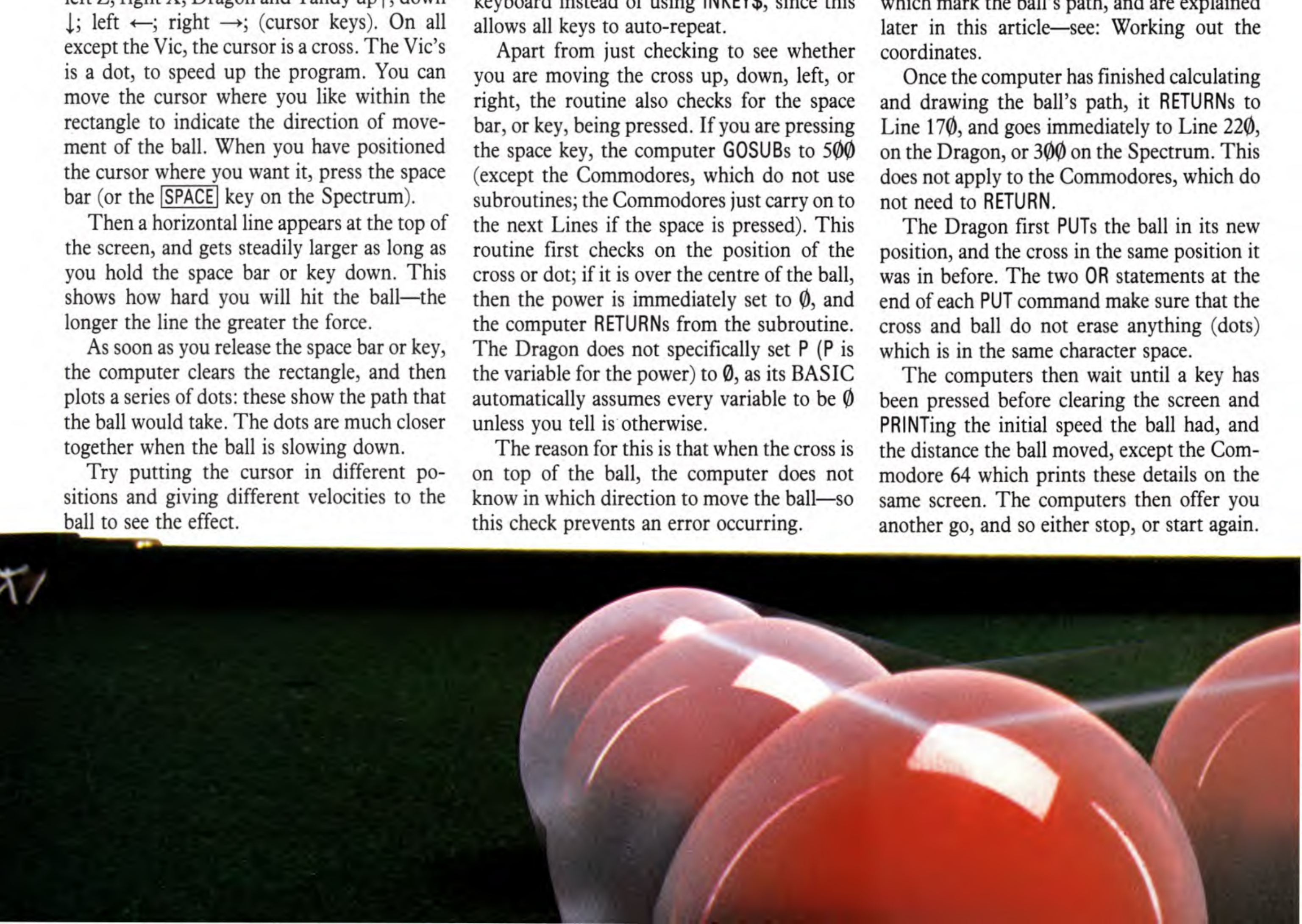
As soon as the condition in Line 540 is no longer true, that is as soon as the space bar/key is released, the computer goes onto the next lines.

These first delete the ball, which was still in its last position. The Spectrum uses OVER 1 and the CIRCLE command to erase it, while the Commodore, Dragon and Tandy blank out the whole rectangle. The rest of the program,i.e. the line numbers after 560, involve calculating and printing the dots which mark the ball's path, and are explained later in this article—see: Working out the coordinates.

Once the computer has finished calculating and drawing the ball's path, it RETURNs to Line 170, and goes immediately to Line 220, on the Dragon, or 300 on the Spectrum. This does not apply to the Commodores, which do not need to RETURN.

The Dragon first PUTs the ball in its new position, and the cross in the same position it was in before. The two OR statements at the end of each PUT command make sure that the cross and ball do not erase anything (dots) which is in the same character space.

The computers then wait until a key has been pressed before clearing the screen and PRINTing the initial speed the ball had, and the distance the ball moved, except the Commodore 64 which prints these details on the same screen. The computers then offer you another go, and so either stop, or start again.



The Acorn program starts off by setting the correct MODE, setting the graphics cursor and defining a UDG for the ball. It then calls the PROCedure PROCBLOCK four times to put in the table.

PROCBLOCK draws a rectangle, but in two halves: it first draws a triangle, and then adds another next to it to make a rectangle. This strange way of drawing a rectangle is used as the computer can easily fill in the triangle with the PLOT command. The numbers in brackets after each call to this PROCedure refer to the positions of each block on the screen.

After this, Line 6Ø sets the colour, and Line 7Ø PRINTs the word POWER at the top of the table.

Line 8Ø sets up four variables. BX and BY are the coordinates of the ball, while CX and CY are the coordinates of the cross.

The colour is then changed to Exclusive OR white (Exclusive OR colours were explained on pages 371 to 374), and the ball and cross are put onto the table.

Two more variables are set up in Line 12Ø, NX and NY, which are given the values of the cross coordinates. These new variables are then used to move the cursor around: as the NX and NY coordinates are changed by INKEY commands in Lines 13Ø to 17Ø (when you move the cursor around), the cross is PRINTed at the new coordinates, and, too, at the old. Since the colour is Exclusive OR white, PRINTing over something which is already there effectively erases it.

PREVENTING ERRORS

This routine continues, with the user free to move the cross around with the keyboard until the space bar is pressed. If the space bar is pressed and the cross is on top of the ball, though, the computer stays in the loop: the cross must be separate from the ball, or an error message will result.

Once the computer has left this loop, it calls PROCSHOT. This PROC edure is the heart of the program: it lets you set the power value, shoots the ball, and PLOTs the dots along the path taken by the ball.

The first section, Lines 24Ø to 31Ø, lets you set the power. The power block is rubbed out, and then built up again as long as the space bar remains depressed. Line 31Ø sends the computer back to the start of this loop IF the space bar is still being pressed, otherwise the computer carries on to the next line.

The next line clears the graphics screen, after which the computer carries the same equation and plots the dots in the same way as the other computers (see: Working out the coordinates, below). Lines 52Ø and 53Ø print the distance that the ball travelled, and the initial speed it had, once the ball has finished moving. The ENDPROC in Line 54Ø sends the computer back to Line 17Ø, and a GOTO in that line sends the computer back to Line 9Ø, to start the program again.

As you can see from the diagram on page 587, you can find out where an object will bounce after hitting a side by working out the angle at which the object hit the side to start with.

Unfortunately, the computers do not have any way of directly measuring angles like this, so they use an equation instead.

WORKING OUT COORDINATES

Each of the programs sets up variables for the positions of the ball and the cross (in fact there are two sets of coordinate variables for the cross, but only one is used here). The computer needs another set, too—the difference between the positions of the cross and the ball. Once it has this, it can work out what direction the ball is travelling in to start with.

The variables which hold the "difference" between the position of the ball and of the cross are DX and DY. The Spectrum and Dragon and Tandy set up these variables in Line 56Ø, while Commodore does this in Line 18Ø, and the Acorn program in Line 33Ø.

There are several other variables, too, which the program uses for the calculations. V is a variable set up for the initial speed; its initial value is a proportion of the power rating which you set up (the exact size of this proportion varies between each computer, because of the different screen sizes. This is also true with some of the other variables).

SQUARE ROOTS

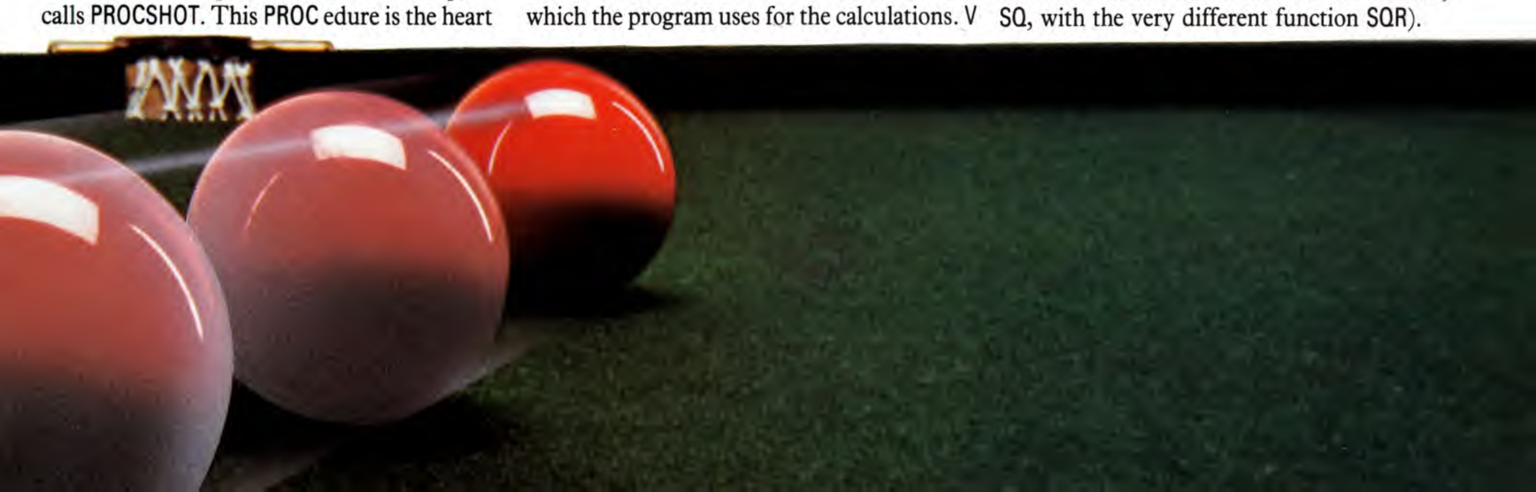
The variable SQ is set equal to the Square Root of (DX*DX+DY*DY). This is actually just working out the distance between the cross and the ball, using a mathematical law: Pythagoras's Theorem, which says that the length of the longest side of a right-angled triangle equals the square root of (the square of both the other sides).

The variable T is the time interval used in the equations. By making the value of T larger, you can make the program faster, although less dots would be plotted. The reason for this is that the total time that the ball travels would be divided by a larger interval, and so the number of steps would be smaller.

As soon as the time interval has been set, the computer plots a point- the first dot of the path. In fact, the Commodore 64 PRINTs the ball—as well as leaving a path of dots.

Then the programs set up another two variables for the calculations. These are the values for speed, in both the x and the y (or the horizontal and the vertical) axes. In other words, VX holds the velocity that the ball is travelling in the horizontal direction, and VY holds the velocity that the ball is travelling in the vertical direction. Both of these variables can hold negative values: if they do, it simply means that the ball is travelling from right to left (for VX) or from down to up (for VY).

The value of these two variables are worked out, as you can see from Line 61Ø (Spectrum, Dragon and Tandy), Line 23Ø (Commodore), or Line 38Ø (Acorn), by multiplying the speed, V, by the variable DX (or DY for VY) and dividing the result by the distance between the centre of the ball and the centre of the cross: the variable SQ (you should be careful not to confuse this variable, SQ, with the very different function SQR).

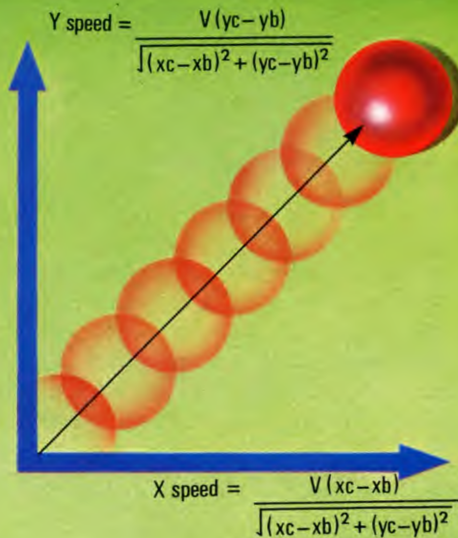


Working out the ball's speed

The programs use several equations to work out the new position of the ball, and its speed. The equations on the diagram separate the speed in the x and the y directions.

The computers solve these two equations for every position in the ball's path, but in a slightly different form from that above. The constants (Xc–Xb) and (Yc–Yb) are replaced in the program by Dx and Dy. With these, the computers calculate the new speeds every move, to take deceleration into account.

As the equation shows, the computer multiplies the difference between the x and y coordinates of the ball and the cursor by the overall speed, and divides this by the result of the bottom line of the fraction: the square root of (the difference between the x coordinates of the ball and the cursor



squared plus the difference between the y coordinates of the ball and the cursor squared).

The next two equations actually work out the new x and y coordinates. They look like this:

Xn = Xb + Vx*T – Ø.Ø1*T²
Yn = Yb + Vy*T – Ø.Ø1*T²

The new position of the ball is calculated by adding a number to the old x and y coordinates. The number to be added is worked out by multiplying the x or y speed by the time, and then subtracting a small fraction of T squared, to take deceleration into account.

The speed is then decreased by a small amount (Ø.1) and the ball's new position plotted, before the loop is run again.

THE BALL'S NEW POSITION

At this point, your computer knows all the values it needs to be able to calculate the new position of the ball—so that is precisely what it does. Line 62Ø, for the Spectrum, Dragon and Tandy, Line 24Ø for the Commodore, and Line 39Ø for the Acorn, calculate the new values for BX and BY (the Commodore 64 calculates new values for the variables LX and LY, since it needs to remember the old values of BX and BY so it can erase the old ball).

The sum that the line works out looks like this for each coordinate:

BX = BX + VX*T – SGN (VX) *.1 *T *T

Although this looks extremely complicated, it is not—once you know what each part does.

The calculations add a number to BX, or BY, to give the new coordinate. The number to be added can be negative, depending on the direction the ball is travelling in.

The first step in the calculation puts in the present value of BX, and then adds the extra distance that the ball has just travelled. This extra distance is worked out in two stages.

To start with, the speed is multiplied by the time interval. This is quite logical: if you think about what speed actually is, it is the distance travelled per unit of time; so, to find out the distance travelled, you simply multiply the speed by the time unit.

Unfortunately, the program also needs to make a slight adjustment for the fact that the speed held by VX and VY is not the actual, overall speed—each is just the speed the ball is travelling *in one direction*. This is the reason for the rather strange-looking calculations at the end of the sum, involving the sign of VX, or VY, and a fraction of T squared.

The function SGN gives the result 1, –1, or Ø, depending on whether the 'argument' (the number in brackets, whose sign you want to know) is positive, negative, or zero. It is used here so that, if the vertical or horizontal speed is negative, instead of subtracting a number from the distance travelled, (which would actually increase the distance travelled if the change in the distance was negative—the change in the distance is negative if it is travelling either upwards, or towards the left, remember).

The computer adds the distance travelled to the right or left to the horizontal coordinate, and then subtracts the value it has just worked out. The final result is the new coordinate. The same calculations are carried out on both coordinates of the ball.

SLOWING DOWN

The program then decreases the speed by Ø.1 and checks to see if the speed is less than Ø.1 (if it is it goes to the end of the routine—which is explained below). It then runs through four lines to check whether the ball would have hit a side on the way to its new position (by using four IF ... THEN statements to check if the coordinates of the new position fall outside the limits of the table). If the ball would have hit a side, then the value of the variable DX (or DY if the side was on the top or bottom of the table) is set equal to minus DX or DY.

The variables BX and BY (LX and LY on the Commodore) are also altered to put the ball's position where it ought to be. The numbers from which the old coordinates are subtracted are exactly double the table's limit. So, the highest coordinate of the table on the Spectrum is 127, which means that whenever the ball hits the top side, of the table, BX becomes 254 – BX (254 is twice 127).

After these checks the computer goes back to do the calculations again for the next point.

The end of the routine (which is reached when the speed is less than Ø.1) is another set of four IF ... THEN statements which check to see if the program is trying to PRINT or PUT the ball on top of the side of the table, before the ball is actually put onto the screen.

Some of the coordinates used in this set of IF ... THEN statements may be different to those in the last set of checks, since this time the thing being put onto the screen is a ball and not a dot—and the ball takes up more space, so the coordinates need to be adjusted in some of the checks.

FURTHER IDEAS

The program can be used in a lot of your own programs. A few logical extensions of it might be to convert the program into a snooker program: you could add some pockets for the ball to fall into, and a scoring routine. You could also use it as the basis for games such as knocking through a wall.

ACORN PROGRAM SQUEEZER-2

- WORKING OUT HOW MANY BYTES HAVE BEEN SAVED
- CONVERTING HEX INTO DECIMAL
- PRINT NUMBERS ON THE SCREEN

The assembly language program in part one of this article stripped unnecessary spaces and REM statements from your BASIC programs. Here, a routine calculates the bytes saved

Apart from its practical function, this is a very useful assembly language routine to understand because it includes a utility that you can use over and over again in other programs—it converts a hex number into decimal and prints it on the screen.



The first thing that you have to do is tie the routines together, so load in the source code from last week. Obviously the first routine must call the one given here, so you should change the RTS in Line 48Ø of the assembly language listing given before to JMP SHOWSAVE. And Line 125Ø which switched off the assembler and closed the pass loop last time has to be overwritten. What follows here is the subroutine SHOWSAVE:

```
124Ø .SHOWSAVE
125Ø SEC
126Ø LDA &72
127Ø SBC &7Ø
128Ø STA &7Ø
129Ø LDA &73
13ØØ SBC &71
131Ø STA &71
132Ø .NUMBER
133Ø LDX #1Ø
134Ø LDA #13
135Ø JSR &FFE3
136Ø .N2
137Ø LDA NUMDATA–1,X
138Ø STA &73
139Ø LDA NUMDATA–2,X
14ØØ STA &72
141Ø LDY #255
142Ø .N3
143Ø INY
144Ø SEC
145Ø LDA &7Ø
146Ø SBC &72
147Ø STA &7Ø
148Ø LDA &71
149Ø SBC &73
15ØØ STA &71
151Ø BCS N3
152Ø LDA &7Ø
153Ø ADC &72
154Ø STA &7Ø
155Ø LDA &71
156Ø ADC &73
157Ø STA &71
158Ø TYA
159Ø CLC
16ØØ ADC #48
161Ø JSR &FFEE
162Ø .N4□DEX
164Ø DEX
165Ø BNE N2
166Ø LDA #13
167Ø JSR &FFE3
168Ø JMP &FFE3
169Ø .NUMDATA
175Ø ]:NEXT
18ØØ FOR T=Ø TO 4
181Ø P%!(T★2)=1Ø↑T
182Ø NEXT
```

The first thing that is done in SHOWSAVE is setting the carry flag. There are some subtractions to be done.

When the microprocessor jumps out of the first part of the stripper, the old program pointer, in 72 and 73, and the new program pointer, in 7Ø and 71, are pointing to a point just past the end of both programs. In fact, they are pointing to what they took to be the high byte of the BASIC line number of the line which doesn't actually exist after the end of the BASIC program.

It's the difference between these two pointers that gives the number of bytes the stripper has saved, so it doesn't matter that these pointers are past the end of the program, because they are both past the end by the same amount.

So the instructions in Lines 125Ø to 128Ø subtract the low byte of the new program counter in 7Ø from the low byte of the old program in 72 and put the result in 7Ø. And the instructions in Lines 129Ø to 131Ø subtract the high bytes and put the result in 71. Done in that order, any borrow from the first subtraction is automatically accounted for in the second subtraction by the carry flag. And since the old program must be longer than, or the same length as, the new, stripped version, there can't be any borrow carried over from the second subtraction.

The number of bytes saved by stripping the program is now stored in 7Ø and 71. All you have to do now is convert this hex number into decimal and print it on the screen.

THE CONVERSION

The theory behind the conversion is quite simple. To work out the fifth place of decimals—that is, the left-hand place, in this case—10,000 is repeatedly taken away from the number, and the program counts how many times it will go. The next place to the right is worked out by repeatedly taking 1000 away and seeing how many times it will go. The hundreds are worked out by taking 100 away repeatedly and counting how many times it will go. The tens are worked out by taking 10s away. And, for completeness, the ones are worked out by taking 1 away repeatedly and counting how many times it will go.

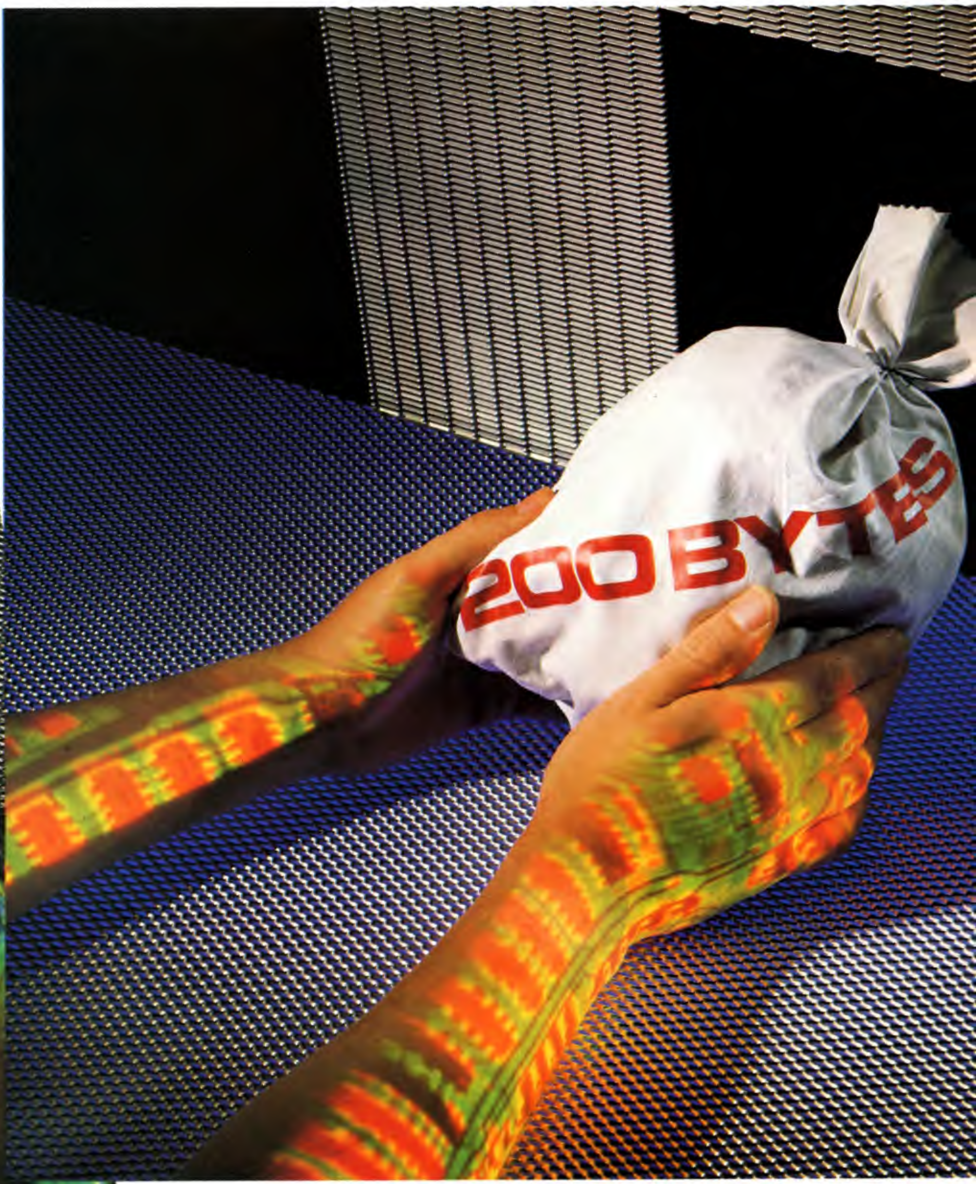
These numbers—the 1ØØØØ, 1ØØØ, 1ØØ, 1Ø and 1—are stored in a table labelled NUMDATA at the bottom of the assembly listing by the FOR ... NEXT loop at line 18ØØ to 182Ø.

The instruction in Line 133Ø loads the X register with 1Ø. This is going to be used as an index pointer to keep your place in the NUMDATA table, which is composed of five words or ten bytes.

The next instruction loads the accumulator with 13 and JSR &FFE3 jumps to the subroutine at FFE3. This routine gives a carriage return/line feed when the contents of the accumulator are 13. This is only done to leave a line space between the last line on the screen and the screen display of the bytes saved.

The instruction on Line 137Ø then loads the accumulator with the high byte of the factor from the NUMDATA table. This is stored in zero-page memory location 73. And the instruction on Line 139Ø picks up the low byte. That's stored in 72.





LDY #255 loads the Y register with FF. The Y register is going to be used as the counter, and to start it off, FF is loaded into it so that when it hits the INY instruction in Line 143Ø it begins the cycle with the value Ø.

The instruction in Line 144Ø sets the carry flag again, ready for some more subtractions. This time the value picked up from the table and stored in the 72 and 73 is taken away from the hex value of the number of bytes saved. Again this is done low byte first, then high byte—so that any borrows are automatically taken into account. The results are stored back in 7Ø and 71, ready for the next subtraction.

If the second subtraction does not require a borrow, the carry flag will be set. Remember, this works the opposite way around from what you'd expect—a borrow clears the carry, no borrow sets it. So if the number from the table can be taken away, the branch is made back to N3 in Line 142Ø by the instruction BCS—Branch on Carry Set. The counter in the Y register is then incremented and the whole subtraction process is gone through again.

The processor will go round and round this loop, taking away the number from the table and incrementing the Y register until the subtraction won't go. When this happens, it will require a borrow, the carry flag will be cleared and the processor will proceed to the next instruction.

The problem is that, by this point, the subtraction will then have been performed one too many times. So the instructions in Lines 152Ø to 157Ø add what's been subtracted on the last pass back on again. You'll note that there is no CLC—CLear Carry flag—instruction here, even though you are about to add. This is because the carry flag must already be clear, otherwise the BCS instruction would have sent the processor back to N3 again.

In Line 158Ø, the TYA transfers the contents of the Y register—that is, the counter that's been clocked up—into the accumulator. The carry flag is then cleared by the CLC on Line 159Ø.

This is not entirely necessary as the TYA instruction does not affect the carry flag and the addition before then cannot have overflowed. You were just adding the last subtraction back on again, remember. But still, the cautious programmer puts clear carry flag instructions before additions and set carry flag instructions before subtractions whenever they occur.

The number in the accumulator must be between 1 and 9. The stripper cannot have saved more than 28160 bytes, so each cycle cannot have clocked up more than 9 or it would have been accounted for in the cycle before.

48 is added to this number to give the ASCII value. And the subroutine starting at FFEE finds the pixel pattern for this character, prints it on the screen and moves onto the next character square.

The instructions in Lines 163Ø and 164Ø DEcrement the X register, twice. This moves the pointers in Lines 137Ø and 139Ø onto the next two bytes in the NUMDATA table. And the BNE in Line 165Ø will branch back to the beginning of this routine again, in Line 136Ø, to work out the next decimal digit working from left to right.

If the end of the table has been reached and the 1s have been dealt with, the second DEX will give Ø. So the BNE N2 instruction does not operate and the processor moves on to the instruction in Line 166Ø. This loads the accumulator up with 13 again and JSR FFE3 puts a carriage return/line feed at the end of the line.

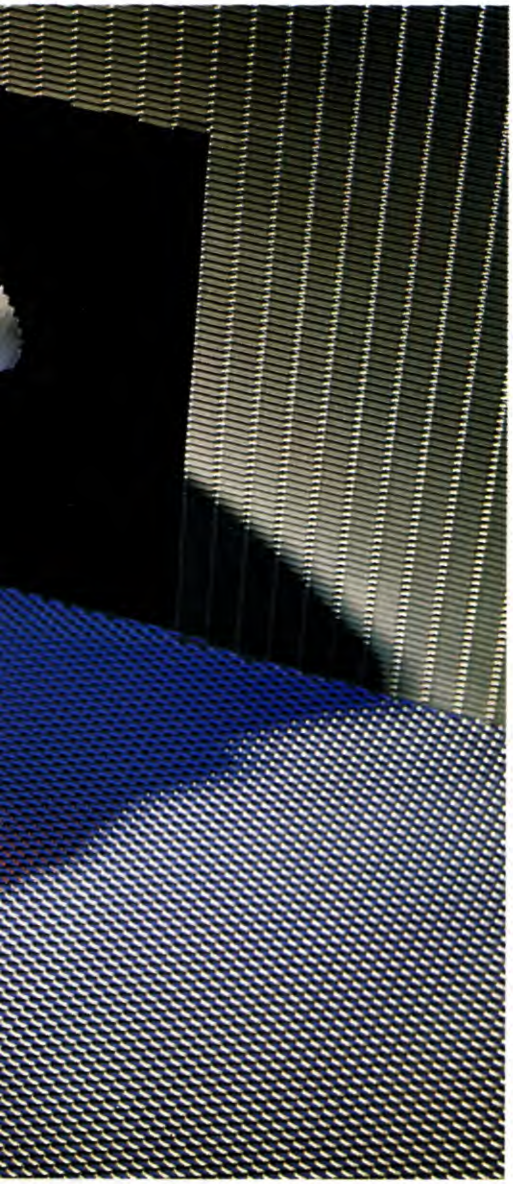
The subroutine that starts at FFE3 has an RTS built into it, otherwise it wouldn't return to the program after it was called. So when, in Line 168Ø, the processor jumps to it, it gives another carriage return/line feed—in other words, it gives another line space—then, when it hits the RTS, it returns to BASIC.

Again, the closed bracket on Line 175Ø, switches the assembler off. And the NEXT closes the FOR ... NEXT loop required to assemble the program. The switch-off assembler instructions from part one were overwritten, remember.

SAVEING, LOADING, CALLING

Once you have RUN this as a BASIC program to assemble it, you SAVE and LOAD it in exactly the same way as you did with the basic stripper. Only, you have to work out the number of bytes you need to save again. In this example you would use ★SAVE "STRIP" Ø900□ØA25.

The same instruction is used to CALL the routine as the second routine is accessed out of the first. Remember to set A% to give you the required stripping.



Q+A

How should I use the zero page on my BBC Micro?

Some of the BBC Micro's zero page is set aside for system variables. So if you are working with one—the pointer to the beginning of the BASIC program area, say—you need to copy it into a free user area of the zero page. The BBC User Guide gives details of the processor's use of the zero page. But locations 7Ø to 8F in hex are free for user routines.

The reason for transposing the value of a pointer into a user zero page location is that, if you tried to manipulate the number in the system variable location it would be in constant danger of being overwritten by the operating system.

So the initial value of pointers are picked up and copied into the user section of the zero page. The advantage of using the zero page is that the associated instructions take one less byte of memory. It is always more efficient to use zero page locations for commonly used values.

DRAGON/TANDY PROGRAM EDITING

All micros have an edit facility and the Dragon system is one of the most sophisticated. But this can make it confusing to use and hard to exploit to the best advantage

Unless you've been very careful, and probably more than a little lucky, RUNning a program for the first time will almost certainly throw up an error message. Or problems might well be found once you've eradicated the errors, and you try to test the program properly.

Debugging a program can be a time-consuming process—see pages 334 to 338—particularly if each offending line needs to be re-entered from scratch. All microcomputers have some kind of editor to speed up alterations to programs, but the Dragon and Tandy system is particularly comprehensive, if a little complicated. Here's how to use it to the best possible effect.

Suppose you have just found an error in Line 2Ø of your program, for example. To bring the editor into play type EDIT 2Ø and [ENTER]. The line will appear towards the bottom of the screen. Underneath it will appear the line number, with the cursor next to it. The screen will have something like this, in which part of the bracket in Line 2Ø is missing:

```
2Ø X=RND(1
2Ø
```

The machine is now in edit mode, and you can make alterations to the line you've chosen. If you have made a mistake and told the machine to enable you to edit the wrong line, just press [ENTER], or [E]—for exit—and the machine will drop out of edit mode. Now type EDIT followed by the correct line number, then [ENTER].

Once the correct line is ready to be edited, there is a range of commands which you can give the editor—and it's at this point that the confusion can start. The commands are very largely single keystrokes, or numbers with single keystrokes.

In the example given above, you want to extend the line by one bracket. Just tap [X]—eXtend—and the cursor will jump to the end



- GETTING THE MOST FROM THE DRAGON/TANDY EDITOR
- STARTING EDITING
- LEAVING EDIT MODE
- CHANGING CHARACTERS
- INSERTING CHARACTERS
- DELETING CHARACTERS
- EXTENDING LINES
- CUTTING OFF LINES
- SEARCHING FOR CHARACTERS

of the line, something like this:

```
20 X=RND(1
20 X=RND(1
```

You can now type the missing bracket, and tap [ENTER] to take you out of Edit mode.

A more complicated case is when you want to insert a number in a line like this:

```
50 X=0:Y=0:W=363
```

EDIT Line 50. This displays the line number and the flashing cursor on the screen, and you now have to find the position where the character or characters have to be inserted. You can move the cursor in three ways. Either tap the space bar until the cursor appears at the right position; or tell the machine how many spaces to move the cursor by typing the number of moves and tapping the space bar; or you can use S to tell the machine to Search for the first occurrence of a particular character.

In Line 50, let's say you left out a 2—X should be equal to 20. The cursor needs to be positioned over the zero. Either tap the space bar twice, or type 2 [SPACE], or S0—the cursor will appear over the right place. Now tell the machine that you want to insert. Type I for Insert and it will allow you to make your additions. The additions are typed normally, followed by [ENTER] if you don't want to make any more alterations.

On the other hand, if you have a further error, say if Y should have been equal to 100, you will want to stay in edit mode—but you can't move the cursor until the insert is cancelled. Type [SHIFT]↑ and you will be able to move the cursor forward using the space bar without a number of useless spaces appearing. Move the cursor to the second zero (using one of the three methods) and tap [I]. You can now add the missing 10.

Sometimes you'll want to change things rather than add to them. Here's another short example.

```
70 PRINT "SGEVE WOZNIAK"
```

EDIT the line, and position the cursor over the offending G. Now press [C]—Change—and you can change the letter under the cursor. If you want to change more than one letter you can type a number before [C]. If you want to change three consecutive letters, for example, you'd type 3C.

Deleting a letter, or series of letters is easy, too. Make sure you are in edit mode, position the cursor over the letter you wish to delete and type [D]. The letter will disappear, and the space will be closed up. You can delete a string of letters by typing the number of letters you wish to delete followed by [D]—e.g. 6D will delete SGEVE *and* the following space if the cursor is put over the S.

Another problem might be a line which is too long, such as this one:

```
90 X=89:Y=76:PRINT "TIME FLYS LIKE AN
   ARROW, FRUIT FLIES LIKE A BANANA"
```

It would be quite understandable if you wanted to rid your program of the PRINT statement! Go into Edit mode, and move the cursor to where you want to delete from. Type [H]—Hack—and the rest of the line will disappear. The machine will be left in Insert mode, so you can either add a different ending to the line, or press [ENTER].

There is another way of removing the end of a line, using [K]—Kill. [K], alone, will kill up to the end of a line, whilst if you type 3KD, the line will be killed up until the third D.

When you have been doing a number of alterations to a line, it's sometimes helpful to know how the line stands—typing [L] will list the current state of the line.

If you've made a mistake with your editing, you can return to the line's original state by typing [A]. [Q] does exactly the same thing, but exits from Edit mode, too.

If you want to practice with the editor, try putting this program right:

```
10 DEF FND(X)='X'P
20 CLS
30 PRINT"NU,MBER","CUBE"
40 FOR A=1 TO 13 STEP2
50 PRINTA,FNC
60 NEXT
```

It should finish looking like this:

```
10 DEF FNC(X)=X*X*X
20 CLS
30 PRINT"NUMBER","CUBE"
40 FOR A=1 TO 13
50 PRINTA,FNC(A)
60 NEXT
```

Altering the program could involve using nearly all of the EDITing commands. Here's one solution to the problem that you might like to try out on your machine:

```
EDIT 10 SD CC 4[SPACE]IX[SHIFT]↑SP
   CX ENTER
EDIT 30 S,D ENTER
EDIT 40 4[SPACE]H ENTER
EDIT 50 X(A) ENTER
```

Summary of EDIT commands

[L]	Lists the current state of the line
[C] character	Changes character
n [C] characters	Changes the next n characters. Value of n must be keyed
[I]	Inserts characters
[D]	Deletes character
n [D]	Deletes n characters
[H]	Hacks—deletes—rest of line, and goes into Insert mode
[X]	Extends the line, and goes into Insert
[S] character	Searches for first occurrence of character
n [S] character	Searches for nth occurrence of character
[K]	Kills as far as nth occurrence of character
n [K] character	Kills—deletes—as far as nth character
n space	Advances the cursor by n spaces—the default is one space
n ←	Backspaces n spaces
shift ↑	Returns to Edit mode from Insert mode
[ENTER]	Leaves Edit and Insert modes
[E]	Leaves Editor
[A]	Returns to original state
[Q]	Returns line to original state and leaves Editor

OVER TO THE BANKER

To complete the game of Pontoon, you have to program the computer to take its turn. In addition, there's a description of the rules in case you are not familiar with the game

Before you start on the remainder of the programming, it's worth a recap on the rules of the game.

Pontoon is played with a standard 52-card pack of playing cards. The cards from 2 to 10 count as face value, the picture cards count as ten and ace counts as 1 or 11, according to the needs of the player. In this game, the computer keeps track of the total for you.

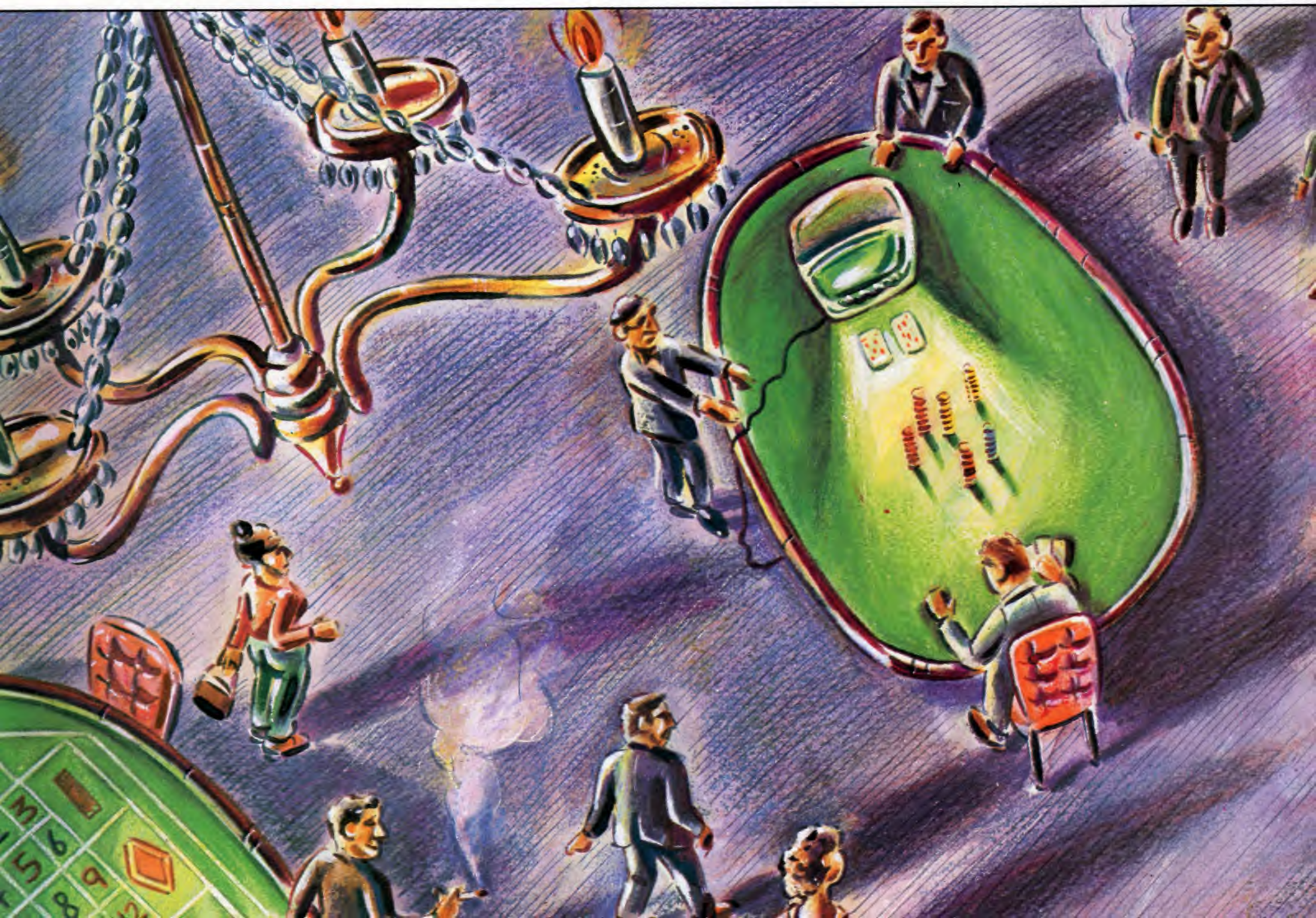
The game is usually played with money, or chips, but in this computer version the computer needs to be programmed to hold the score in imaginary chips—more unscrupulous programmers could make the machine less than honest with the chips! The computer will be programmed to play as banker at all times and look after the deal.

At the start of the game, the cards are shuffled and two cards are dealt, face down. The screen display will show the player's card face up, but the program has been designed so that the machine doesn't know what cards the player is holding. The player must now lay a bet on this first card, before another card is dealt to both the player and the dealer.

The object of the game is to finish with a better hand than the banker's—that is, a higher total value. A hand which adds up to over 21 is bust, and loses. A hand with a score between 16 and 21 beats the banker *only* if the machine is holding a lower hand, or busts. There are two special hands—pontoon, which consists of an ace and a ten or picture card, adding up to 21 in two cards; and a five-card trick, which is any hand of five cards which total 21 or less. The player's pontoon beats anything held by the banker, except pontoon itself. A five-card trick will also beat anything held by the banker, except pontoon and another five-card trick.

HOW THE PROGRAM WORKS

After the player has received a second card, the program checks if pontoon is being held. If it isn't, there are three options which may be open to the player. Firstly, if the player has cards totalling 16 or over, and is satisfied with the cards, he may elect to 'stick'. The turn will end, and the game proceeds to the banker's play. Secondly, if the total is below 16, or the player isn't satisfied with the total, more cards



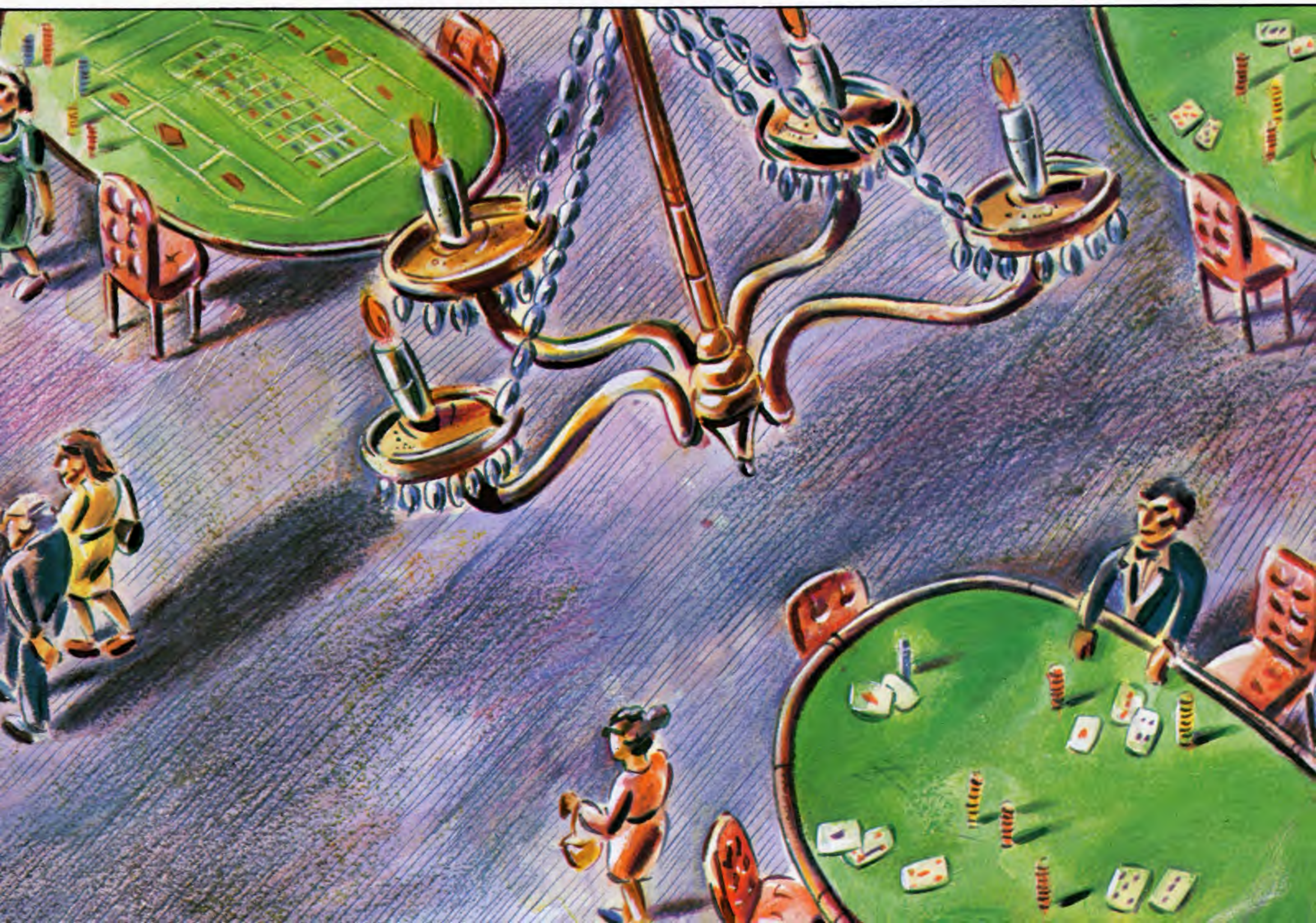
may be acquired by either 'twisting' or 'buying'. In normal pontoon, buying a card entails the player placing another stake equal to the original one. In exchanges the banker deals the card face down. In the computer version, there is no distinction made between cards being dealt face up and face down, so the card appears at the side of the previous cards. Twisting doesn't entail the player adding to the stake, and in a game with real cards, the card would be dealt face up—here it makes no difference. The player cannot buy after twisting, and if trying for the last card of a five-card trick, the player must twist. The program checks after each card that the player, or banker, hasn't exceeded a total of 21 and therefore gone bust.

If the player has bust, the banker takes all the stake, and so the computer doesn't bother with its play. Otherwise, the banker turns the two cards over and has to decide whether to stick, or deal more cards. Pontoon at this stage is an automatic win and no more cards are dealt. If there isn't a pontoon, the banker will carry on dealing extra cards until satisfied with the total, or a five-card trick is reached, or the hand busts.

If the banker sticks at a number below 21, and hasn't accumulated a five-card trick, a message appears on the screen: DEALER CALLS ... a number one larger than the score on the cards. If the score is 21, the message PONTOONS OR FIVE-CARD TRICKS ONLY, is displayed. If the banker gets a five-card trick, the player is told PONTOONS ONLY. In a normal game, the players would then have to declare their hands to the banker in the event of a win, but in this game, of course, the computer is already holding the player's score.

If the player has scored a number equal to the announced score, he wins, but if he has less, loses his stake. When the player wins, the banker matches his stake, so that the player receives twice the total stake money. Unlike some sets of rules, no extra chips are paid on five-card tricks and pontoons.

Unlike the normal rules, the deal cannot pass to the player but must remain with the computer—in normal play the bank passes on when pontoon is scored. Here, the computer



is always banker, so it has an advantage at all times during the game—the banker always has an advantage in normal Pontoon, too. The player, then, has the difficult task of breaking the bank—which here means accumulating more than 1,000 chips—to win. The player starts the game having 100 chips, and must not lose them all, or will have lost the entire game.

Load in the last two parts of the program then type in these lines for the complete pontoon game:

```
2500 PAUSE 50
2510 LET DPF=B: LET AF=B: LET X=C:
    LET Y=10
2520 FOR J=C TO 2
2530 LET Z=O(J): GOSUB 5500
2540 IF VA>10 THEN LET VA=10
2545 FOR K=C TO 2: LET W(K)=W(K)+VA:
    NEXT K
2546 IF VA=C AND AF=B THEN LET
    W(2)=W(2)+10: LET AF=C
2550 LET X=X+6: NEXT J
2560 IF W(2)=21 THEN PRINT PAPER 2; INK
    7;AT 14,18; "□PONTOON!□"; AT 15,17;
    "□DEALER WINS□": LET DPF=C:
    GOTO 2000
2600 IF PF=C THEN GOTO 2720
2610 PAUSE 75
2630 IF W(C)>21 THEN PRINT PAPER C; INK
    7;AT 21,B; "□DEALER BUST!□";: GOTO
    2740
2635 IF W(2)>21 THEN LET W(2)=W(C)
2640 IF W(2)<16 THEN GOTO 2800
2650 IF FF=C THEN GOTO 2800
2660 LET PR=(W(2)-8)/13
2670 IF PR<RND THEN GOTO 2800
2700 IF W(2)>21 THEN LET W(2)=W(C)
2710 IF W(2)=21 THEN PRINT AT 20,B;
    "DEALER PAYS PONTOONS
    AND□□□□□□□□□FIVE-CARD
    TRICKS ONLY": GOTO 2000
2720 PRINT PAPER 2;AT 21,B; "DEALER
    PLAYS□";W(2)+C;
2725 IF S(2)>21 THEN LET S(2)=S(C)
2730 IF W(2)>=S(2) THEN PRINT PAPER 2;
    "□AND WINS□": GOTO 2000
2740 PRINT PAPER 2;"□ – YOU WIN□":
    LET CP=CP+BET*2: GOTO 2000
2800 LET Z=C(CC): GOSUB 5500
2805 IF VA>10 THEN LET VA=10
2810 FOR K=C TO 2: LET W(K)=W(K)+VA:
    NEXT K
2820 IF VA=C AND AF=B THEN LET
    W(2)=W(2)+10: LET AF=C
2822 IF W(1)<22 AND X=25 THEN PRINT
    PAPER 2; INK 7;AT 21,B; "□FIVE-CARD
    TRICK!□": GOTO 2000
2825 GOSUB 7000
2830 LET X=X+6: GOTO 2610
```

After a one-second pause in Line 2500, the program starts on the dealer's turn. Line 2510 sets the dealer pontoon flag, and the ace flag to zero, and sets up the coordinates of the corner of the dealer's first card.



THE DEAL

The two previously stored cards are dealt by the FOR ... NEXT loop between Lines 252Ø and 255Ø. Line 253Ø displays the card. Line 254Ø sets the value of the picture cards to ten before Line 2545 adds the dealt card to the running total—two totals in case there's an ace present. Line 2546 adds ten to W(2) if one of the cards is an ace and also sets the ace flag. The next card's position is adjusted in Line 255Ø.

Line 256Ø checks for a pontoon and announces that the dealer wins if there is one. The dealer pontoon flag is set. If the player has pontoon—Line 26ØØ—then the program jumps to the routine which handles the outcomes—who's holding what, and who's won.

Following a pause in Line 261Ø, Line 263Ø checks if the dealer holds more than 21. The message DEALER BUST! appears if the machine is holding more than 21. Line 2635 checks if the higher of the two totals—when an ace is present—has exceeded 21. The program then concentrates on the lower of the two totals. If the dealer holds less than 16, another card is dealt by Line 28ØØ.



ANOTHER CARD?

Lines 266Ø and 267Ø decide if the dealer wants another card. This is not done according to a set rule. It's not a good idea to impose a strict limit on when the dealer will stick, because the player will soon understand exactly what needs to be done to beat the machine. Instead, some kind of random factor has to be introduced (as is done here), so that it's impossible to predict at what number the machine will stick.

When trying to write this kind of routine the problem is that you must make the computer behave a little like a human player. Think how you would play the game—you would be less likely to ask for another card if you were holding 20 than if you were holding 16, because the odds against busting are far worse. By comparing PR with a random number, there is both a random factor and the correct weighting according to the score being held by the dealer.

THE TOTAL SCORE

Line 27ØØ swaps W(2) for W(1) if W(2) exceeds 21. If the dealer gets 21, the message DEALER PLAYS PONTOONS AND FIVE-CARD TRICKS ONLY is displayed by Line 271Ø. If the dealer gets a score below 21, Line 272Ø displays DEALER PLAYS a figure one larger than the actual score.

Line 273Ø compares the dealer's score with the player's score. If the dealer wins, Line 273Ø displays a message, but if the player wins, Line 274Ø displays a message. If the player wins, the chip total is added to in Line 274Ø.

The final section of the program—Lines 28ØØ to 283Ø—is concerned with giving the dealer cards from the pack.

Line 28Ø5 sets the value of the picture cards to ten. The value of the card is added to W(1) and W(2) in Line 281Ø. Adjustments for an ace are made in Line 282Ø, before Line 2822 checks for a five-card trick. The card pointer is adjusted by calling the subroutine at Line 7ØØØ, and the next card's position is calculated by Line 283Ø.

Now add these lines to your pontoon program. They are mainly concerned with the banker's turn:

```
390 IF PL=2 AND(TU=21 OR(TU+
   T3=21ANDCT(1)=Ø))THEN 75Ø
41Ø IF PL=2 THEN 71Ø
71Ø FOR DE=1 TO 5ØØ:NEXT DE
72Ø IF NU=2 AND T(1)>21 THEN 8ØØ
73Ø IF RND(1)<(21-TU)/13 OR TU<16
   OR CT(1)=1 THEN 51Ø
74Ø IF T3<>Ø AND RND(1)<(21-
   (TU+T3))/13 THEN 51Ø
75Ø PRINT "■■■I'LL STICK...":
   GOTO 8ØØ
78Ø IF PL=2 THEN S1$="I":
   S2$="MY"
8ØØ T(PL)=TU:IF TU+T3<22 AND
   TU+T3>TU THEN T(PL)=TU+T3
81Ø IF PL=1 OR (PL=2 AND CT(2)=1)
   THEN 84Ø
82Ø IF T(2)<21 THEN PRINT
   "■■■■DEALER CALLS";T(2)+1
83Ø IFT(2)=21ANDPO(2)=ØTHEN
   PRINT"■■■■PONTOONS OR FIVE-
   CARD TRICKS ONLY!"
84Ø FOR DE=1 TO 1ØØØ*PL:NEXT DE:IF
   PL=1 AND T(1)>21 THEN 86Ø
85Ø NEXT PL
86Ø CH$="CHIPS":IFTB=1 THEN
   CH$="CHIP"
87Ø IFCT(2)=1AND(PO(1)=Ø OR
   CT(1)=1)THENPRINT"■■
   YOU'VE LOST■"TB;CH$:
   GOTO9ØØ
875 IF(CT(1)=1 OR PO(1)=1)AND
   PO(2)=Ø THEN 89Ø
88Ø IF(T(2)<22 AND T(2)=>T(1))
   ORT(1)>21THENPRINT"■■
   YOU'VE LOST■"TB;CH$:GOTO9ØØ
89Ø PRINT"■■YOU'VE WON■"TB*2;
   CH$:MU=MU+TB*2
9ØØ IF MU<1 OR MU>999THEN PRINT
   "■AND THE GAME!":GOTO93Ø
91Ø PRINT "■■"E2$:FOR DE=1TO
   3ØØØ:NEXT:IF PO(1)=1 OR PO(2)=1
   THEN 6Ø
92Ø GOTO 76Ø
93Ø PRINT "■■■■ANOTHER GO.
   (Y/N)?"
94Ø GET A$:IF A$="Y" THEN RUN
95Ø IF A$<>"N" THEN 94Ø
96Ø PRINT "■":END
```

PL2 is player 2—the banker. If the banker has 21 he will decide to stick. Line 39Ø, then, checks whether the computer is holding 21, and if so jumps to Line 75Ø which announces I'LL STICK. Line 41Ø makes the computer miss out the player's turn lines.

Line 71Ø is the start of the computer's turn proper. There's a pause before a series of checks. Line 72Ø checks for two aces, whilst Line 73Ø decides if the computer wants to take another card. If the running total is less than 16, or if the player has already accumulated a five-card trick the machine must take another card.

TO STICK OR NOT TO STICK?

The first part of the line—the part using the random mathematical expression—is quite interesting. It's there to make the computer behave something like a human player. The easiest way to make the computer play would be to make it always stick at the same number—say, 19. The problem is that the player will soon realise how to beat the machine, and will always try to beat 19. By incorporating the randomness into the IF ... THEN, the computer becomes impossible to predict, but at the same time it should be stopped from twisting too often at higher scores which would make it bust in too many cases. By subtracting TU from 21, dividing by 13 and then comparing with the random number, the machine can be made more likely to twist at 16 than 20.

AN ACE IS DEALT

Line 74Ø does a similar job to Line 73Ø, but looks after the case when an ace has been dealt. If neither of the two lines makes the machine receive another card, Line 75Ø tells the player that the machine is sticking, and jumps to Line 8ØØ.

S1$ and S2$ are changed so that they are right for the machine—player 2, remember—by Line 78Ø.

After either player has stuck, Line 8ØØ makes sure that, if an ace has been dealt, the score on which play was halted is the higher of the two alternatives. If it is the player's turn, or the computer has a five-card trick, the program uses Line 86Ø to set CH$ to CHIPS or CHIP. Line 85Ø closes the FOR ... NEXT loop which you set up in Line 76Ø in the previous part of Games Programming.

WIN OR LOSE

If the player has lost, Line 87Ø displays how many chips have been forfeited. If the player has won, Line 875 makes the program jump to Line 89Ø which tells the player that he has won, and calculates how many chips are now being held. Finally, Line 88Ø is the other condition where the player loses. If the player loses or wins, Line 9ØØ checks if the number of chips have dropped below 1, or gone above 1,000, and if it has, announces AND THE GAME! Next, Line 93Ø asks if another game is wanted—Lines 93Ø to 96Ø are just an 'another go?' routine.

If the player or the machine has scored a pontoon, Line 91Ø ensures that the cards are shuffled ready for the next hand. If no pontoon is being held, the program doesn't shuffle the cards, and the top two are dealt by returning to Line 76Ø.

There's very little now to add to the Acorn program:

```
25Ø IF M/2+2>P*T THEN PRINT
   "YOU HAVE□";:VDU 67,72,69,65,
   84,69,68,13,1Ø:END
33Ø FIN=N:F=TT:TT=Ø:TT2=Ø:
   N=Ø:BF=Ø:TP=1ØØ:PP=1ØØ
35Ø E=Ø
36Ø PROCTWIST:FOR T=1 TO 2ØØØ:
   NEXT:PROCTWIST
37Ø IF ((RND(1)>(21-TT)/13 AND TT>15)
   OR (F=21 AND FIN=2)) AND
   NOT(FIN=5 AND F<22) OR E<>Ø
   THEN 44Ø
38Ø REPEAT
39Ø CLS:PRINT"I'VE GOT□";TT
4ØØ FOR T=1 TO 2ØØØ:NEXT
41Ø PROCTWIST
42Ø IF FIN=5 AND F<22 THEN UNTIL
   (N=5 OR TT>21):GOTO 44Ø
43Ø UNTIL (RND(1)>(21-TT)/13 AND
   TT>15) OR E<>Ø
44Ø CLS:PRINT"I'VE GOT□";TT
45Ø IF E=2 THEN PRINT"I'VE BUST"
46Ø IF E=3 THEN PRINT'"PONTOON"
47Ø PROCSCORE
54Ø IF (F=21 AND FIN=2) OR (N=2 AND
   TT=21) THEN 26Ø
122Ø DEF PROCSCORE
123Ø IF FIN=2 AND F=21 AND TT<>21
   AND N<>2 THEN 13ØØ
124Ø IF FIN=5 AND F<22 AND N=5 AND
   TT<22 THEN 128Ø
125Ø IF FIN=5 AND F<22 AND NOT (N=5
   OR (N=2 AND TT=21)) THEN 13ØØ
126Ø IF F<22 AND F>TT THEN 13ØØ
127Ø IF TT>21 AND F<22 THEN 13ØØ
128Ø PRINT"YOU LOST□";BET;
   "□CHIPS"
129Ø GOTO 131Ø
13ØØ PRINT"WELL DONE YOU WON":
   M=M+BET*2
131Ø ENDPROC
```

Line 25Ø is a bit of fun to stop the player tampering with the amount of chips given at the beginning of the program.

THE COMPUTER'S GO

Line 33Ø sets up some variables and clears some others which were used during the player's turn. Line 36Ø turns the dealer's two cards over. Line 37Ø decides if the dealer should ask for another card. The line itself is quite interesting because the random expression enables the computer to play something like a human.

The easiest way for you to program the computer's turn would be to say that the



computer will stick at any number over 19, for example. The snag with this is that the player will soon understand what the computer is doing and will be able to beat it—or at least, know how to beat it. Introducing the randomness means that the computer becomes unpredictable, but that doesn't mean that it will play stupidly. By comparing 21 – TT with the random number, the computer can be made less likely to stick at 16 than 20, and will behave something like a human player, sometimes making some rash twists.

Line 34Ø also checks that the dealer has over 15 before allowing a stick, also that there isn't a pontoon or that E indicates a five-card trick or bust. If the machine doesn't stick, the program continues to the REPEAT ... UNTIL loop between Lines 38Ø and 43Ø.

First, the dealer's score is displayed. There's a pause before the next card is turned over by Line 41Ø. Five-card tricks are checked by Line 42Ø. A random decision like



that in Line 37Ø is made again in Line 43Ø. Messages are displayed on screen by Lines 44Ø to 46Ø as they are needed. Line 47Ø calls the scoring PROCedure, which is to be found between Lines 122Ø and 131Ø.

Before you add these lines, delete : GOTO 19Ø from Line 65Ø.

```
500 GOSUB4000:GOTO540
510 CX=CX+50:GOSUB3500:
    NC=NC+1
520 FORK=1TO1700:NEXT
530 IF DL>21 THEN 620
540 DT=DL+10*(DA AND(DL<12)):
    IFDT=21 AND NC=2 THEN CLS:
    PRINT"□DEALER HAS PONTOON":
    PF=1:GOTO610
550 IF NC=5 THENPRINT"□DEALER
    HAS A 5-CARD TRICK":GOTO 610
560 R=20-DT:IF RND(100)<R*R*R
    OR DT<16 ORP5=1 THEN510
570 CLS:IF P5=1 THEN PRINT
    "□YOU HAVE A 5-CARD TRICK" ELSE
    PRINT "□YOU HAVE";PT
580 PRINT"□DEALER PLAYS";:
    IFDT<21 THEN PRINTDT+1 ELSE
    PRINT"□PONTOONS AND□□□
    □□□□5-CARD TRICKS ONLY"
590 FORK=1TO500:NEXT:IF P5=1 THEN
    630
600 IFDT<PT THEN630
610 PRINT"□DEALER WINS":
    GOTO 690
620 CLS:PRINT"□DEALER BUST. YOU
    WIN":GOTO 640
630 PRINT"□YOU BEAT THE DEALER"
640 MN=MN+2*BT:GOTO 700
660 GOSUB4000:IF DL=11 THEN
    PRINT:PRINT"□BUT SO DOES THE
    DEALER":GOTO610
670 GOTO 630
690 PRINT:IF MN<1 THENPRINT
    "□YOU'VE LOST ALL YOUR
    MONEY":GOTO790
700 IFPF=1 THENPRINT:PRINT
    "□DEALER IS SHUFFLING THE
    CARDS□□□FOLLOWING THE
    PONTOON": GOSUB 1500: GOTO 750
710 NF=N-1:IF NA>N THEN NF=N+51
720 FORX=NF TO NA+1 STEP-1:
    Q=RND(X-NA)+NA-1:T=SQ(X):
    SQ(X)=SQ(Q):SQ(Q)=T:NEXT:
    IF NA>N THEN 740
730 FORX=0TO9:SQ(X+52)=SQ(X):
    NEXT:GOTO750
740 FORX=0TO9:SQ(X)=SQ(X+52):
    NEXT
790 PRINT:PRINT"□DO YOU WANT
    ANOTHER GO (Y/N)?"
800 A$=INKEY$:IFA$<>"Y" AND
    A$<>"N" THEN 800
810 IF A$="Y" THEN CLS:GOTO 170
820 END
3500 SCREEN1,1:GOSUB1000:
```

```
    GOSUB2000:IF NM=1 THEN DA=1
3510 IF NM>10 THEN DL=DL+10 ELSE
    DL=DL+NM
3520 RETURN
4000 NN=N:N=D1:CX=6:CY=108:
    GOSUB3500
4010 N=D2:CX=56:GOSUB3500: N=NN
4020 FORK=1TO2000:NEXT:NC=2:
    RETURN
```

At this stage in the game, the player's cards will be arranged across the screen. In addition, there will be the backs of the dealer's cards. The first part of the dealer's go, then, will be for the two cards to be turned over—Line 500 calls the subroutine between Lines 4000 and 4020. To turn the cards over, the subroutines concerned with printing front faces are called—initially by calling the one starting at 3500, which turns the high resolution screen on and calculates the running total. The other card is displayed by calling the subroutine again, but this time after line 4010 has changed the X coordinate of the card. The card number is again adjusted. The subroutine ends at Line 4020.

Line 510 changes the X coordinate again and calls the subroutine at Line 3500. The dealer gets the next card. Line 520 is a pause, before Line 530 checks if the dealer has bust—Line 620 tells the player that the dealer has bust. Line 540 checks if the dealer has pontoon and sets the pontoon flag if he has—a message appears on the text screen, too. Line 550 detects a five-card trick.

TO STICK OR NOT TO STICK?

Probably the most interesting part of the program is in Line 560. The line is concerned with how the computer decides when to ask for another card. It would be all too easy to program a simple strategy into the machine so that it always sticks when it reaches, say, 19 or above, but it would be so easy to beat because the player would soon understand the way that the computer is playing. Instead, some kind of randomness needs to be built into the program so that the computer becomes unpredictable. At the same time you must be careful not to make its tactics suicidal—it shouldn't twist too often when at 20.

A variable, R, is set up in Line 560. The total is subtracted from 20. Next, a random number between 1 and 100 is chosen, and compared with R*R*R. If the random number is less, or the running total is less, than 16, or the player already has a five-card trick, then the dealer asks for another card. The way R has been chosen means that the chance of asking for another card decreases as the running total increases—only very seldom will the machine ask for another card when at 19, but is most likely to ask for another card when at 16.

THE SCORE

Line 570 displays the player's score, or indicates a five-card trick. Line 580 displays the dealer's play, or announces PONTOONS AND FIVE-CARD TRICKS ONLY.

If the player is holding a five-card trick and the dealer isn't, Line 590 tells the player YOU BEAT THE DEALER, after a short pause. Similarly, Line 600 compares the dealer's and player's scores and displays the outcome using Line 610 or Line 630.

If the player has won, Line 640 adds what the player has won to the chips he already has. If the player or dealer has won with a pontoon, the pack has to be shuffled. Line 700 checks the pontoon flag.

Line 660 follows the announcement that the player has pontoon. It is the case when both the dealer and the player have pontoon. The outcome is that the dealer wins—Line 610. Line 670 sends the program back to Line 630, YOU BEAT THE DEALER.

If the number of chips has fallen below zero, Line 690 displays the message YOU'VE LOST ALL YOUR MONEY.

If the pontoon flag has been set, Line 700 tells the player of the pontoon and shuffles the cards. Lines 710 to 740 are a small shuffle to simulate the dealer gathering up the cards.

Finally, there's a 'Do you want another go?' routine from Line 790 to Line 820.

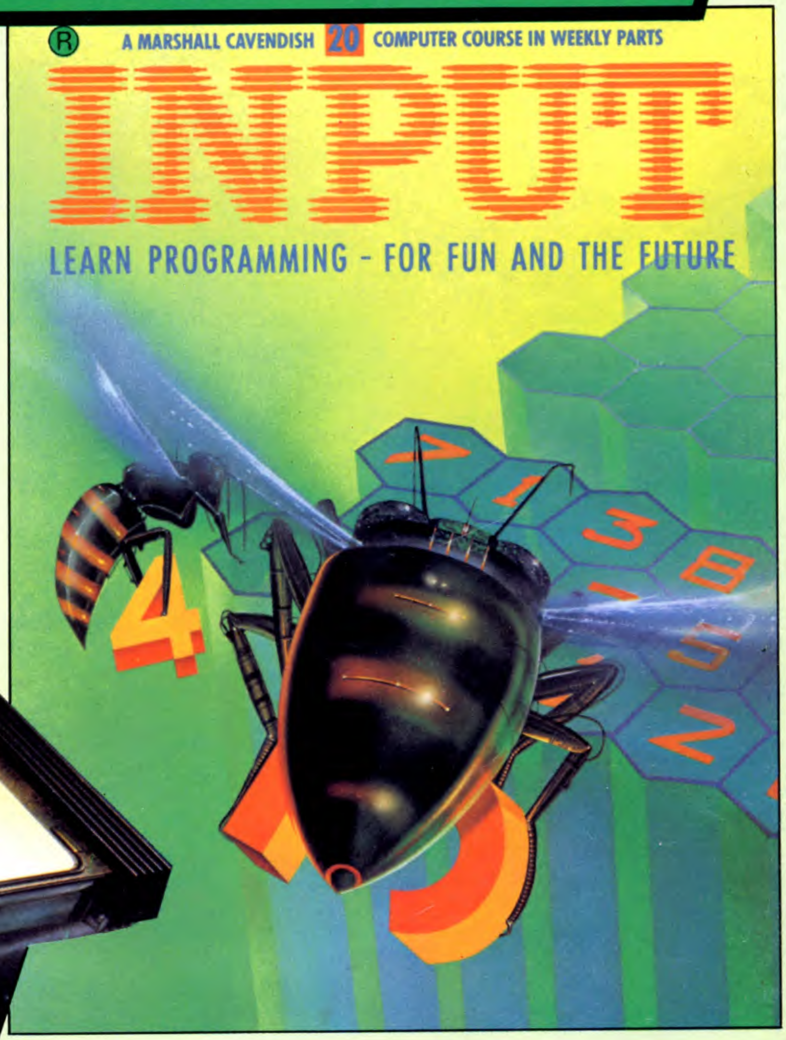


CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 20...

- Now get things into PERSPECTIVE with WIREFRAME DRAWING
- For really broad horizons, linking your computer to the world, via a MODEM
- Understand more about how a computer can CREATE AND USE FILES
- For better adventure games, try a TEXT COMPRESSOR PROGRAM
- PLUS ... For SPECTRUM users, an automatic CONVERTER FOR MICRODRIVES







ASK YOUR NEWSAGENT FOR INPUT